CARDIFF
UNIVERSITY
PRIFYSGOL
CAERDYDD

IMPROVING THE BEES ALGORITHM FOR COMPLEX OPTIMISATION PROBLEMS

A thesis submitted to the Cardiff University

In candidature for the degree of

Doctor of Philosophy

By

Sameh Otri, B.Eng., M.Sc.

Manufacturing Engineering Centre

School of Engineering

Cardiff University

United Kingdom

2011

DECLARATION AND STATEMENTS

DECLARATION

This work has not previously been accepted in substance for any degree and is not concurrently submitted in candidature for any degree.

Signed ………………………… (candidate) Date ………………

STATEMENT 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of ………………………… (insert MCh, MD, MPhil, PhD etc, as appropriate)

Signed ………………………… (candidate) Date ………………

STATEMENT 2

This thesis is the result of my own independent work/investigation, except where otherwise stated.
Other sources are acknowledged by explicit references.

Signed ………………………… (candidate) Date ………………

STATEMENT 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ………………………… (candidate) Date ………………

STATEMENT 4: PREVIOUSLY APPROVED BAR ON ACCESS

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loans **after expiry of a bar on access previously approved by the Graduate Development Committee.**

Signed ………………………… (candidate) Date ………………

In the name of Allah,

the Most Merciful, the Most Kind

DEDICATION

THIS THESIS IS DEDICATED TO

MY GRANDPARENTS,

MY PARENTS,

MY IN-LAWS,

MY WIFE BAYAN,

AND MY CHILDREN ABDULLAH, JUDE AND JANA.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who have helped me, in any way, to successfully complete my PhD thesis.

In particular, I would like to express my sincere thanks to my supervisor, Prof. D.T. Pham for his encouragement, invaluable advice and guidance throughout my study.

I owe my parents, brothers and sisters a great gratitude for all their love and support. Special love goes to my beloved mother, Nadrah, may Allah be mercy with her.

This work would not have been possible without the support, appreciation and patience of my wife Bayan and our children Abdullah and Jude.

ABSTRACT

An improved swarm-based optimisation algorithm from the Bees Algorithm family for solving complex optimisation problems is proposed. Like other Bees Algorithms, the algorithm performs a form of exploitative local search combined with random exploratory global search. This thesis details the development and optimisation of this algorithm and demonstrates its robustness.

The development includes a new method of tuning the Bees Algorithm called Meta Bees Algorithm and the functionality of the proposed method is compared to the standard Bees Algorithm and to a range of state-of-the-art optimisation algorithms.

A new fitness evaluation method has been developed to enable the Bees Algorithm to solve a stochastic optimisation problem. The new modified Bees Algorithm was tested on the optimisation of parameter values for the Ant Colony Optimisation algorithm when solving Travelling Salesman Problems.

Finally, the Bees Algorithm has been adapted and employed to solve complex combinatorial problems. The algorithm has been combined with two neighbourhood operators to solve such problems. The performance of the proposed Bees Algorithm has been tested on a number of travelling salesman problems, including two problems on printed circuit board assembly machine sequencing.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

2-Opt	Two (paths) Optimal
ABC	Artificial Bee Colony
ACO	Ant Colony Optimisation
ACS	Ant Colony System
ATSP	Asymmetric Travelling Salesman Problem
CACO	Continuous Ant Colony Optimisation
CAP	Component Allocation Problem
CIAC	Continuous Interacting Ant Colony
CPU	Central Process Unit
CSP	Component Sequencing Problem
DNA	Deoxyribonucleic Acid
EA	Evolutionary Algorithms
EP	Evolutionary Programming
ES	Evolutionary Strategy
FAP	Feeder Arrangement Problem
Gas	Genetic Algorithms
MBTD	Moving Board with Time Delay
MMAS	Max-Min Ant Colony System
NE SIMPSA	Stochastic Simulated Annealing Optimisation Procedure
PAP	Pick and Place Machine
PC	Personal Computer
PCB	Printed Circuit Board
PID	Proportional Integral Derivative
PSO	Particle Swarm Optimisation
RAM	Random Access Memory
SA	Simulated Annealing
SI	Swarm Intelligence
SIMPSA	Deterministic Simplex Method

SMT	Surface Mount Technology
SQP	Sequential Quadratic Programming
TS	Tabu Search
TSP	Travelling Salesman Problem

NOMENCLATURE

CHAPTER 2

p_{ij}	Probability of moving from node i to node j
τ_{ij}	Pheromone intensity of the corresponding link, (i, j)
η_{ij}	Desirability of the corresponding link, (i, j)
α	Pheromone amplification coefficient
β	Heuristic amplification coefficient
φ	The global pheromone evaporation parameter
$\Delta\tau_{ij}$	The amount of pheromone deposited
q_0	An exploitation/exploration parameter
ρ	Local pheromone decay parameter
τ_0	The initial value of the pheromone trails
L_{nn}	The length of tour produced by NNH
L^{best}	The length of the global best tour
T^{ib}	The iteration best tour
Cl	The length of the candidate list

CHAPTER 3

q	Recruitment strategy parameter
f(x)	Objective function
N	The number of scout bees
M	The number of sites selected
E	The number of top-rated (elite) sites
Nep	The number of bees recruited for the best e sites

Nsp	The number of bees recruited for the other (m-e) selected sites
Ngh	The initial size of each patch
$f_i(X)$	Objective functions
maxF	Maximum value of the function F
$minF$	Minimum value of the function F
X	The column vector

CHAPTER 5

μ	Mean value of the process variable being monitored
σ	Standard deviation of the process
$d(c_i,c_j)$	The distance between cities *i* and *j*
F_1	A feeder
H	An assembly head
F_2	Multiple feeders
D_{Total}	The total distance between all visited cities (components)
V	The average speed of movement of the table or assembly head
T_{Total}	The cycle time
N	The total number of components
T_i	The time required to place the component C_i
G	The number of assembly heads positioned between the pick-up and placement heads on the turret
t_1	The time for the table to move from the location of component C_{i-1}
C_0	The starting position of the table
x_i	x coordinate of the component c_i
y_i	y coordinate of the component c_i
v_x	The velocity of the x-y table in the x direction

v_y	The velocity of the x-y table in the y direction
t_2	The time for the feeder array to change the pick-up feeder from the feeder supplying component C_{i+g} to that supplying component C_{i+g+1}
C_{i+g+1}	The component that is to be picked-up when component C_i is placed onto the PCB
C_N	The last component to be assembled onto a given PCB
x_i	The x coordinate of the feeder f_i
y_i	The y coordinate of the feeder f_i
v_f	The speed of the feeder carrier
N	The number of cities in a TSP

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

The challenges faced by industry today to increase efficiency, especially in relation to the use of expensive resources within tighter time constraints, have posed difficult issues for engineers. These challenges have been met with new approaches to processing embodied in new optimisation techniques as traditional optimisation techniques are no longer adequate.

New intelligent optimisation algorithms have emerged in the field of artificial intelligence, many of them inspired by nature. One such algorithm is the Bees Algorithm which mimics the foraging behaviour of honeybees.

The Bees Algorithm has been initially applied to a range of continuous problems only. Also, it has not yet been tested for dynamic problems. Finally, it has a large number of parameters that need to be tuned to produce good results. However, the tuning normally has to be carried out manually, which can be a laborious process.

The main motivations for the research presented in this thesis were:

1. To improve the Bees Algorithm, enabling it to address the above-mentioned issues. The improvement includes a new method of automatically tuning the Bees Algorithm called the Meta Bees Algorithm.

2. To demonstrate the robustness and efficiency of the new algorithm in comparison to existing algorithms.

3. To ascertain the applicability of the algorithm to challenging optimisation problems by implementing it for a Travelling Salesman Problem, a Printed Circuit Board problem and a stochastic optimisation problem.

1.2 RESEARCH AIM AND OBJECTIVES

The overall aim of this work was to prove the hypothesis that the new Bees Algorithm is able to solve complex optimisation problems faster than other optimisation techniques.

The following objectives were set to achieve this aim:

1. Survey current swarm-based optimisation algorithms, including the Bees Algorithm.

2. Develop a new method of evaluating the fitness function of the Bees Algorithm to accommodate non-deterministic types of problems.

3. Implement new local search operators into the neighbourhood search process with the aim of reducing the number of parameters needed to run the Bees Algorithm.

4. Apply the proposed optimisation tools to different categories of continuous and combinatorial optimisation problems.

5. Validate the different versions of the proposed algorithm by applying them to different benchmark optimisation problems and compare the results obtained with those of other optimisation methods.

To achieve the above objectives, the following methodology was adopted:

- Review of previous work: an extensive survey was performed of the state of the art in swarm-based optimisation techniques, focusing on bees-inspired algorithms, to identify research trends and potential solutions.

- Algorithm development and evaluation: the standard Bees Algorithm was extended by adding a new evaluation method for the fitness function of both local and global search parts of the Bees Algorithm. The performance of the new version of the algorithm was assessed by running it on a number of benchmark problems. The results obtained were compared with those of other optimisation techniques to test the effectiveness of the proposed method.

1.3 THESIS ORGANISATION

The remainder of the thesis is organised as follows: Chapter 2 reviews the background literature on swarm-based optimisation algorithms relevant to the work presented in the thesis. This covers material on the Evolutionary Algorithms (EA), the Genetic Algorithms (GAs), Ant Colony Optimisation (ACO), Particle Swarm Optimisation (PSO) and bees-inspired algorithms including the Bees Algorithm itself. A number of recent papers concerning the optimisation of the Bees Algorithm are also discussed.

Chapter 3 describes a study of the main characteristics of the standard Bees Algorithm. This is undertaken through an exploration of the parameters of the algorithm in order to help understand the methods by which its performance is improved, such as avoiding premature convergence. The study reveals the implementation of a new method of tuning the Bees Algorithm called Meta Bees Algorithm (a Bees Algorithm within a Bees Algorithm). The results of this study were tested with five benchmark problems and compared with those obtained by other optimisation algorithms.

Chapter 4 describes the use of the Bees Algorithm to solve a stochastic optimisation problem using statistical analysis. The algorithm employed to carry out this task was designed with a new fitness evaluation method based on computing the average fitness value for each bee over a number of trials rather than computing the value of a single trial. The method enabled the algorithm to be applied to those situations where the fitness value changes even when the location visited by a bee remains the same. To test the algorithm, the parameter value optimisation of a metaheuristic method is employed. This test took the shape of parameter setting for the Ant Colony Optimisation algorithm which is employed to solve a Travelling Salesman Problem (TSP).

Chapter 5 studies the applications of the Bees Algorithm to combinatorial problems. The development of the Bees Algorithm for the TSP serves as an illustrative example for such applications and provides a platform to demonstrate the characteristics of the proposed algorithm. The chapter also discusses those features of the Bees Algorithm employed in solving more complex combinatorial optimisation problems and their application to facilitate the optimisation of a solution to two problems associated with a Printed Circuit Board (PCB) assembly line, namely, component sequencing and feeder arrangement.

Chapter 6 summarises the main contributions of this work and the conclusions reached and provides suggestions for future work.

CHAPTER 2

OPTIMISATION ALGORITHMS AND THEIR APPLICATIONS

This chapter introduces optimisation techniques and algorithms and some terms and definitions employed in optimisation. It briefly reviews the state of the art optimisation algorithms employed in this work for comparison with the Bees Algorithm. The chapter emphasises two main algorithms, namely those inspired by ants and honeybees. How they function both in nature and in engineering practice is looked at. The application of these two algorithms to different types of problems is also considered.

2.1 OPTIMISATION

Optimisation seeks to find the “best” solution to a problem and it also studies algorithms or methods applied to solve that problem.

2.2 OPTIMISATION PROBLEMS

Each optimisation problem consists of four essential components: an objective function or fitness function to be optimised, a set of variables that need to be calculated to find the value of the objective function(s), a set of constraints that determine the allowed values of the variables, and the search space that encompass all possible solutions to a problem. With regards to these four components:

1. The degree of nonlinearity of the objective function determines whether the problem solved is a linear or nonlinear problem. In addition, if the criteria of the optimisation problem can be expressed in one objective function it is called a single-objective problem, otherwise, in a multi-objective problem a number of objective functions are needed.

2. The type of variables employed divide problems into either continuous problems, or discrete and combinatorial problems must be considered. In continuous problems the variables employed in the objective function are real values, whereas in discrete and combinatorial problems they are restricted to assume only discrete values.

3. If the problem has no constraints or conditions that satisfy it, it is called an unconstrained problem, otherwise it is called a constrained problem where it contains one or more constraints that must be satisfied.

4. The search space determines if the problem is a static/deterministic problem which does not change over time, or if it is a dynamic/stochastic problem where the search space changes over time (Blackwell and Branke, 2004).

2.3 OPTIMISATION ALGORITHMS

The theory and applications of optimisation algorithms have recently been developed rapidly in the field of artificial intelligence and the following provides a brief description of the various elements of these algorithms (Eberhart et al., 2001).

1. On the types of searches applied to solve the optimisation problem there are two possibilities: Single Point Search (Trajectory) (SPS) which is also known as a Direct Search (DS), and Population-Based Search (PBS) which is also known as a Swarm Based Search (SBS). With SPS the algorithm generates a single solution. Whereas with PBS, a strategy is employed 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 reduces the value of the objective or cost function.

2. On the number of solutions generated, there are two possibilities: a Single Optimum Solution (Single Objective) and a Multi Objective Optimisation/Multiple Optimal Solution.

3. On the search space of candidate solutions algorithms have employed two methods: Exploration (Diversification) also known as a Global Search and Exploitation (Intensification) also known as a Local Search. Local Search algorithms exploit neighbourhoods while Global Search algorithms explore the entire search space.

4. On the accuracy of generated solutions (Time/accuracy), three outcomes are possible (Laporte, 1992):

- Exact algorithms are methods which utilise mathematical models and try to find an optimal solution if it is allowed to complete their execution and they try to prove that the solution obtained is actually an optimal one. Unfortunately these types of techniques are time consuming. Some instances in the exact methods category are Branch and Bound (Wiener 2003) and Integer Linear Programming (Rego and Glover, 2002).

- Approximate/Heuristic Solution: This provides high quality solutions in short computation time but the algorithm does not guarantee finding an optimal solution and may fall in local optima missing the true optimum solution. Approximation algorithms make use of certain heuristics and iterative improvements to the problem solving process. The approximation algorithms can be further divided into two groups: constructive heuristics and improvement heuristics. Instances in constructive heuristics include Nearest Neighbourhood, Greedy Heuristics and Insertion Heuristics (Rosenkrantz et al., 1977).

- Metaheuristic algorithms: These are general-purpose techniques for guiding and modifying problem-specific constrictions or local search heuristics. They consist of concepts that can be employed to define heuristic methods and they can be applied to different optimisation problems with relatively few modifications. They can produce solutions beyond those that are normally generated in a search for local optimality. Procedures based on evolutionary approaches, Tabu Search, Simulated Annealing, and Multistart strategies fall into this category. Hybrid procedures based on metaheuristic frameworks are also considered as metaheuristic algorithms (Glover and Kochenberger, 2003).

5. In the way the algorithm builds a solution, there are two methods:

o The first is known as constructive, where the algorithm generates solutions from scratch by adding solution components systematically (step by step). They are speedy and return reasonably good solutions but not always guaranteed. An example of this algorithm is the Greedy Constriction Heuristic (GCH).

o The second is known as Improvement (Local Search), where the algorithm improves the current solution by movements to neighbouring solutions. Iterative Improvement Algorithms (IIAs) use this method to find a better solution and replace the current one. However it may get stuck in a poor quality local optimum. An example of an IIAs is the Two (two paths) Optimal (2-Opt) algorithm (Okano et al., 1999) that will be described in detail in chapter five of this thesis. There are algorithms that employ both techniques, they initially use the constructive method then follow it up with the Improvement method such as the Ant Colony Optimisation (ACO). Instances in improvement heuristics include k-opt (Chandra et al., 1999), Lin-Kernighan Heuristics (Aarts et al., 1988; Helsgaun, 2000), Simulated Annealing (Knox, 1994), Tabu Search (Freisleben and Merz, 1996), Evolutionary Algorithms (Merz and Freisleben, 1997; White and Yen, 2004), the Ant Colony System (Stützle and Hoos, 1997) and the Bee System (Lucic and Teodorovic, 2003; Sato and Hagiwara, 1997).

6. In the method used to improve solutions, algorithms have employed two techniques:

o Deterministic and Stochastic: with the first approach, random elements are not employed whereas with the second approach random elements are employed such as the pheromone value used in the Ant Colony Optimisation (ACO). This approach will be explained later in the chapter discussing ACO.

7. Algorithms can also be classed according to where they draw their inspiration from. Of the most recent are a group of algorithms which get inspiration from natural systems such as social, ecological, biological, physical and chemical systems. Two examples of nature-inspired algorithms are Artificial Intelligence (AI) and Swarm Intelligence (SI) (Bonabeau et al., 1999).

8. Search History: Memory-less algorithms use the current state of the search process to determine the next action. Other algorithms incorporate a

memory of the past history of the search such as recently performed steps and the generated solutions.

9. Encoding solutions: Algorithms, like the Bees Algorithm, encode solutions with real-value variables when solving continuous problems, while the same algorithm encodes solutions with discrete variables when solving discrete or combinatorial problems.

After this introduction to essential definitions and terms in the field of optimisation for both algorithms and problems, a review of current metaheuristic algorithms is presented.

2.3.1 Neural Networks (NNs)

In the field of Artificial Intelligence (AI), Artificial Neural Networks (ANNs) are composed of simple interconnecting elements called artificial neurons. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections (weights) between elements and thus the neural network can be trained to perform a particular function by adjusting the values of these connections (Pham and Liu, 1995).

The ANN in general consists of three layers; an input layer connected to a hidden layer, which is connected to an output layer. Each layer comprises of a set of vectors. The weights on the connection between the input vectors and the hidden vectors determine the activity of each hidden layer. ANNs are trained by comparing the output and the target of the network until they are matched.

Neural networks have been trained to perform complex functions in various fields of application including pattern recognition, identification, classification, speech, vision and control systems.

There are many types and structures of ANNs such as Feed-forward neural networks (Kennedy and Eberhart, 1995; Pham and Sholedolu, 2008), Spiking Neural Networks (Pham and Sahran, 2006) and Learning Vector Quantisation (LVQ) (Pham et al., 2006a).

Learning Vector Quantisation (LVQ)

The LVQ neural network was developed by Kohonen (Kohonen, 1989) and has been successfully employed for many classification problems. Figure 2.1 shows an LVQ network, which consists of three layers of neurons: an input layer (buffer), a hidden layer and an output layer.

The network is fully connected between the input and hidden layers and partially connected between the hidden and output layers, with each output neuron linked to a different cluster of hidden neurons (also known as

Kohonen neurons). The weights of the connections between the hidden and output neurons are fixed at 1. The weights of the input to hidden neuron connections form the components of "reference" vectors, with one reference vector assigned to each hidden neuron. When an input vector is supplied to the network for recognition, the hidden neuron whose reference vector is closest in terms of Euclidean distance to the input vector is said to win the competition against all the other hidden neurons to have its output set to "1". All other hidden neurons are forced to produce a "0". The output neuron connected to the cluster of hidden neurons that contains the winning neuron also emits a "1" and all other output neurons, a "0". The output neuron that produces a "1" gives the class of the input vector, each output neuron being dedicated to a different class.

The learning method is supervised (Jain and Dubes, 1988) and based on "competitive" learning, in which neurons compete to have their weights updated. During learning, the neurons in the hidden layer compete amongst themselves in order to find the winning neuron whose weight vector is most similar to the input vector (Kohonen, 1990). The winning neuron gives the class of the input vector. Only the winning neuron will modify its weights using a positive or negative reinforcement learning formula, depending on whether the class indicated by the winning neuron is correct or not. If the winning neuron belongs to the same class as the input vector (the classification is correct), it will be allowed to update its weights and move slightly closer to the input vector (positive reinforcement). On the contrary, if the class of the winning neuron is different from the input vector class (the classification is not correct), it will be made to move slightly further from the input vector (negative reinforcement).

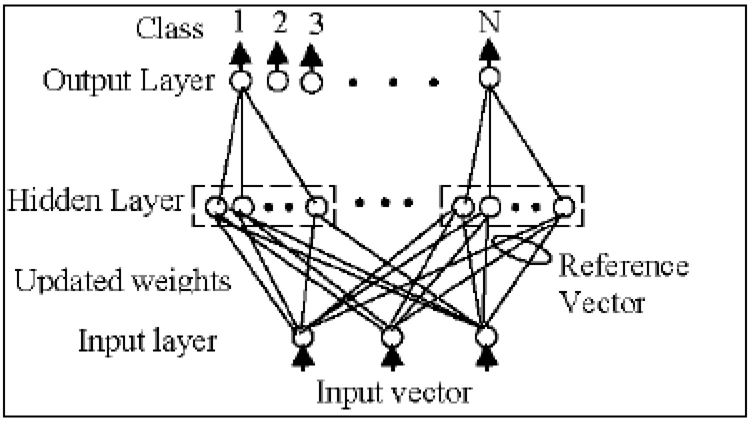


Figure 2.1 Topology of an LVQ Network

LVQ Network Training Procedure

The training of an LVQ network can be regarded as 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 (Pham and Oztemel, 1992). Training proceeds by presenting to the network a pattern of known class taken randomly from the training set. If the class of the pattern is correctly identified by the network, the error component associated with that pattern is null. If the pattern is incorrectly identified, the error component is set to 1.

The procedure is repeated for all the patterns in the training set and the error components for all the patterns are summed to yield the value of the error function for an LVQ network with a given set of reference vectors.

2.3.2 Simulated Annealing (SA)

Annealing is the process of heating up a material and then cooling slowly until it crystallises which allows the metal to achieve a better crystal structure which is more stable and hard-wearing. Heating up the atoms of this material increases their energies and these energies give these atoms a great deal of freedom in their ability to restructure themselves. As the temperature is reduced the energy of these atoms decreases. Ideally the temperature should be deceased at a slower rate. If this cooling process is carried out too quickly many irregularities and defects will be seen in the crystal structure. A slower fall to the lower energy rates will allow a more consistent crystal structure to form.

Simulated annealing was developed by Scott Kirkpatric in the mid 1970's to simulate the actual process of annealing (Pham and Karaboga, 2000). Simulated annealing begins at a very high temperature where the input values are allowed to assume a great range of random values. As the training progresses the temperature is allowed to fall. This restricts the degree to which the inputs are allowed to vary. This often leads the simulated annealing algorithm to a better solution.

2.3.3 Tabu Search

Tabu Search (TS) is a metaheuristic algorithm that uses a local or neighborhood search procedure to iteratively move from a solution *x(i)* to a solution *x(i+1)* in the neighborhood of *x(i)*. TS explores new areas of the search space by modifying the neighborhood structure of each solution as the search progresses until some stopping criterion has been satisfied. It uses a short-term memory structures to determine the new solutions. A Tabu list contains the solutions that have been visited in the recent *n* number of previous solutions. These solutions will be excluded in the search unless one of these solutions is better than the recently-discovered best solution (Pham and Karaboga, 2000). The TS can be employed for solving combinatorial optimisation problems, such as the travelling salesman problem (TSP).

2.3.4 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are stochastic search algorithms that are inspired by the metaphor of natural Darwinian biological evolution. 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. The methods employed in EAs have been described by several researchers (Michalewicz, 1996 and Goldberg, 1989). General to all EAs is also a selection pressure mechanism that removes poor individuals from the population, thus allowing better individuals to monopolise the evolutionary process. EAs also modify the individuals to refine the population of candidate solutions.

Four different implementations of the EAs led to the following four techniques. The following describes the different EAs.

2.3.4.1 Evolutionary Strategies

In the 1960s, Rechenberg and his colleagues were the first to apply the principle of Darwinian evolution to the study and design of engineering systems and modern technology (Rechenberg, 1965). Standard Evolutionary Strategies (ES) use a population of individuals then apply mutation, recombination, and selection operators in order to evolve (evaluation the evolution) iteratively better and better solutions (Baeck et al., 1991).

2.3.4.2 Evolutionary Programming

Evolutionary Programming (EP) (Fogel et al., 1966) traditionally employed a representation tree to develop automata recognising strings in formal languages. However, it was only ten years later that EPs gained worldwide popularity following the creation of Genetic Algorithms (GAs) by Holland (1975).

2.3.4.3 Genetic Algorithms

Genetic Algorithms (GAs) are in many ways very similar to Evolution Strategies (ESs). However, the original applications for which GAs and ESs were developed are different. While ESs were applied first to continuous parameter optimisation problems associated with laboratory experiments, GAs were designed to solve discrete or integer optimisation problems.

In its basic structure, a GA utilises three separate operations for generating a new solution. These operations are population selection, recombination and mutation. Candidate solutions in the traditional GA are encoded in binary bit strings (`chromosomes') for integer and decision variables (Goldberg, 1989). While continuous control variables are approximated and rescaled by equivalent integer variables. A GA initially selects its population randomly then it applies the principle of `survival of the fittest'. A GA attempts to construct new improved solutions by combining the features of good existing ones in a procedure called *crossover.* To maintain diversity within the population, a GA runs a mutation operator that changes the bit value, in the case of a binary coding, from 0 to 1 and vice versa. A GA evaluates the function(s) of the problem as a fitness value for each member of each population to assess that particular population. Mutation rate and crossover rate are two essential parameters required to be tuned carefully.

2.3.5 Particle Swarm Optimisation

The natural flocking and swarming behaviour of birds studied by Craig Reynolds in the late 80s inspired Russel Ebenhart and James Kennedy to introduce the PSO algorithm as a recognised and suitable technique (Kennedy and Eberhart, 1995).

PSO consists of a number of individuals referred to as particles. Each particle in a PSO has a position and a velocity. These particles are attracted to positions in the search space that have high fitness. Each particle has a memory function that remembers two pieces of information, the first piece of information results from the memory of the particle of its past states as the best-so-far position that it has visited, called the *local best*, and the second piece of information results from the collective experience of all particles as the global best position attained by the whole swarm, called the *global best*. Both the local best position of each particle and the global best position of the entire swarm guide the movements of all particles towards new improved positions and eventuality to find the global minima/maxima.

PSO has been applied to many problems such as the training of Feed-forward neural networks (Pham and Sholedolu, 2008) and Clustering (Omran et at., 2005).

2.3.6 Ant Colony Optimisation (ACO)

ACO is one of the most successful metaheuristic algorithms inspired by the foraging behaviour of real ant colonies and was proposed by Dorigo and colleagues for the solution of combinatorial optimisation problems (Dorigo et al., 1996). The collective trail laying and trail following behaviour of ants enable them to find the shortest path from the food source to their nest.

2.3.6.1 Foraging Behaviour in Ants

Ants start their journey for food from their nest by exploring the surrounding area randomly. When food is found, some of it will be carried back to the nest by the ants which will also lay a chemical substance called pheromone on the ground while walking back. The amount of pheromone trails laid by ants recruits more ants to choose the same path to the food source and guides others to the nest. When an ant deposits pheromone on a path that is not one of the shortest, the pheromone will evaporate. As time passes the path will not be followed by other ants and this mechanism is called negative feedback. The more ants that follow a certain path, the more pheromone will be deposited and this reinforces the quality of this path. This mechanism is called positive feedback. Hence the ants find the shortest paths between the nest and food sources (Dorigo and Stützle, 2004; Deneubourg et al., 1990).

2.3.6.2 Ant Colony Optimisation Algorithms

The Ant Colony Optimisation (ACO) algorithm is a constructive search algorithm based on the simultaneous exploration of different solutions by a colony of identical ants. All ant-based algorithms use the positive feedback mechanism represented in the trail-laying trail-following behaviour of real ants by reinforcing good solutions or parts of them. The negative feedback mechanism is implemented through pheromone evaporation to avoid premature convergence (Stagnation) and being trapped in local optima.

Ant colony optimisation has been studied thoroughly and many algorithms have been developed such as the Ant System (AS), Max-Min Ant System (MMAS), Ant Colony System (ACS) and others (Engelbrecht, 2005). The major differences between these algorithms are:

- The way the pheromone update is performed

- The management of the pheromone trails

The following summarises one of the successful algorithms of the ACO, namely the Ant Colony System (ACS). This algorithm has been tested and researched the most.

2.3.6.3 Ant Colony System

The Ant Colony System (ACS) is based on four elements that are employed to solve the optimisation problem. These elements are an exploration/exploitation transition rule, a global pheromone trail updating rule, a local update for the pheromone trail, and the use of a candidate list (Stützle and Hoos, 1997).

The first element of the ACS is that ants use an exploration/exploitation decision rule, called the *pseudo-random-proportional* rule, in which an ant k located at city i chooses a city j Є N_i^k to move to using Equation (2.1) (Dorigo and Stützle, 2004).

$$j = \begin{cases} \arg\max_{l \in N_i^k} \{\tau_{il}(t)\eta_{il}^{\beta}(t)\}, & if\ q \leq q_0\ (Exploitation); \\ J, & otherwise(Biased\ Exploration); \end{cases}$$

Equation (2.1)
(Dorigo, 2004)

Where τ_{ij} is the pheromone intensity and it represents the effectiveness of the move from node i to node j as expressed in the pheromone intensity of the corresponding link, $\eta_{ij} = 1/d_{ij}$ is a heuristic value which is the reverse of the distance between the two cities i and j, β is a parameter that determines the relative influence of the heuristic value, N_i^k is the feasible neighbourhood of ant i when at city k.

q is a random variable uniformly distributed over [0,1], q_0 is a tuneable parameter ($0 \leq q_0 \leq 1$), and *J* is a random variable selected according to the probability distribution given by Equation (2.2) (Dorigo, 2004).

$$p_{iJ}^{k}(t) = \frac{\tau_{iJ}^{\alpha}(t)\eta_{iJ}^{\beta}(t)}{\sum_{l \in N_i^k} \tau_{iJ}(t)\eta_{iJ}^{\beta}(t)}$$

Equation (2.2)

(Dorigo, 2004)

Where α is a parameter to control the influence of τ_{ij} (α is equal to 1 when $q \leq q_0$). This decision rule is employed to balance between exploration and exploitation and has a double function. When $q \leq q_0$, the decision rule exploits the knowledge available about the problem such as the heuristic knowledge about distances between cities in the case of TSP problem, and the learned knowledge memorised in the form of pheromone trails. However, when $q > q_0$ it operates a biased exploration of other tours (arcs).

The second element of ACS is where ants perform online step-by-step pheromone local updates to favour explorations of other new solutions (cities in case of the TSP) instead of the best current solution. The local pheromone trail updates are performed by applying Equation (2.3) (Dorigo, 2004).

Equation (2.3)

$$\tau_{ij}(t) = (1-\rho)\tau_{ij}(t) + \rho\tau_0$$

(Dorigo, 2004)

where $0 < \rho \leq 1$ is a parameter governing local pheromone decay. τ_0 is the initial value of the pheromone trails. It was experimentally found that setting $\tau_0 = (n.L_{nn})^{-1}$, where n is the number of cities in the TSP instance and L_{nn} is the length of a tour produced by the Nearest Neighbour Heuristic (NNH), produced good results. The effect of the local update rule is that each time an ant uses an arc (i, j) its pheromone trail τ_{ij} is reduced, so that the arc becomes less desirable for the following ants. This allows an increase in the exploration of arcs that have not been visited and avoids stagnation. Stagnation occurs when pheromone accumulates on a certain path, which is usually a local optimal solution, and more ants keep choosing this path over and over until eventually all ants choose this path and the algorithm prematurely converges to this local optimal solution.

The third element of the ACS is the global pheromone trail update where at the end of an iteration of the algorithm, once all the ants have built a solution, a pheromone trail is added to the arcs (edges) employed by the ant that found the best tour from the beginning of the trial. The rule of this offline global pheromone trail update is shown in Equation (2.4) (Dorigo, 2004).

Equation (2.4)

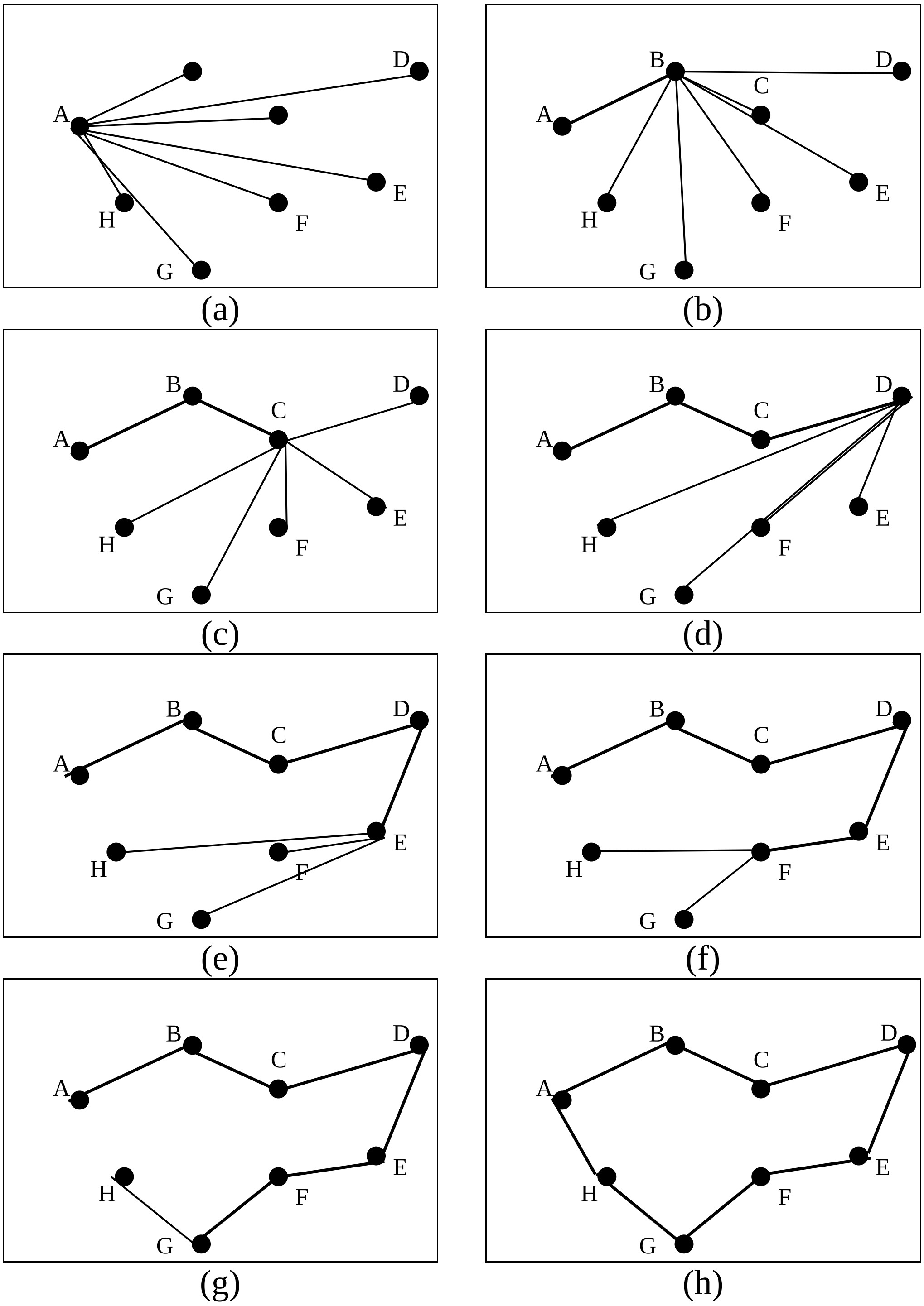
$$\tau_{ij}(t+1) = (1-\varphi)\tau_{ij}(t) + \varphi\Delta\tau_{ij}(t)$$

(Dorigo, 2004)

where τ_{ij} is the pheromone trail level on the edge (ij), φ Є (0, 1) is a parameter governing the global pheromone decay (evaporation), $\Delta\tau_{ij}(t)$ = 1/L^{best} where L^{best} is the length of the global-best tour T^{gb} found since the beginning of the trail or the length of the iteration-best tour T^{ib} found during the current iteration (t).

With the forth element, the ACS exploits a data structure called the candidate list that provides additional local heuristic information and reduces computational time when solving large problems. A candidate list is a list of preferred cities to be visited from a given city. The candidate list of a city contains a number of cities ordered by increasing distance. In the ACS when an ant is in city i, instead of examining all the unvisited neighbours of city i, it chooses a city to move to that is on the candidate list but has not been previously visited by that ant. All visited cities are placed on a list called the Tabu list which is referred to before visiting a city. After visiting all cities on the candidate list, other cities are then examined.

To explain how the ACS operates, the following simple example is employed where the ACS is applied to solve a TSP which is considered a minimisation problem and the candidate solution is defined as a sequence of cities. In this example, there are eight cities (A to H) and for simplicity, it is assumed that ant one is placed in city A, ant two is placed in city B and so on. Every time an ant (k) needs to move from city i to city j, it adds its current location to its Tabu list and then uses Equation (2.1) where it generates a random value for the parameter q, when $q \leq q_0$, ant (k) becomes greedy and exploits the knowledge available about the problem and goes to city (j) which has the maximum product of the amount of pheromone on the edge (ij) and the shortest distance between the two cities. While when $q > q_0$, the ant (k) explores new solutions using a probability decision from Equation (2.2). Each city has a candidate list; its length is defined by the number of cities listed (cl). In this example, cl = 2, where cities (B) and (H) are on the candidate list of city (A) and they will be explored by ant (1) before other cities. In this example, it is assumed that ant (1) moves from city (A) to city (B), then city (B) will be added to the Tabu list to avoid being visited twice by the same ant. After moving from city (A) to city (B), ant (1) updates the pheromone on the link between the two cities using local update Equation (2.3). For the next step, ant (1) again calculates the possibilities of moving from its current city (B) to those cities that are not in its Tabu list (C to H) using the same Equation (2.1) and so on until ant (1) visits all the seven cities as shown in Figure 2.2. The length of the tour made by ant (1) will be calculated by adding the length of the arc between each two cities from the tour. The process will be accomplished by each ant and at the end of the iteration there will be eight tours generated by eight ants. The shortest of these tours will be selected as the best tour and the arcs that form this tour will be updated using the global update formula in Equation (2.4). The ants are placed again randomly for a second iteration. The algorithm goes on until a stopping criterion is met such as the minimum number of iterations or the global tour length has been found.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 2.2 Application of the ACS for a Simple TSP problem

2.3.6.4 ACO and the Speed of Convergence

This section provides an introduction on the effect of the stochastic nature of the optimisation problem on the speed of convergence in metaheuristic methods in general and ACO in particular.

The time and number of evaluations consumed in finding the best solution in metaheuristic optimisation methods depends on two factors. The first factor is controllable and the other is not. The first factor is the right selection of the values of the set of parameters for the metaheuristic method. The other factor is the stochastic nature of these types of methods i.e. the randomness that resides in some parts of the algorithm. An example of this can be seen when the ACS is applied to solve a TSP problem. It is initialised by setting the value of its parameters such as the number of ants (M) and the exploitation/exploration ratio (q_0). Here, the randomness of the algorithm is in the distribution of the ants over the available cities. It should be noted that, for each iteration, these positions are always chosen randomly.

Experiments have been conducted to study the effect of various parameters on the speed of convergence, here are a summary of these experiments.

- A small size of candidate list (cl) decreases the convergence time (Dorigo and Gambardella, 1997).

- A value of q_0 define close to one decreases the convergence time but the quality of solution is lower (Bonabeau et al., 1999).

- Large numbers of ants increase the solution quality as it widens the search space but also increase the computation time to a large extent (Wong and Komarudin, 2008).

- The overall quality of solution is increased when the local pheromone evaporation rate is close to one but it slows down the convergence speed and this leads to a suboptimal solution while if it is close to zero then no cooperative behaviour can emerge (Hao et al., 2006).

2.3.6.5 ACS Parameters Tuning

Research in parameter setting has provided many ways of approaching the problem; some researchers, Adenso-Diaz and Laguna (2006) have studied parameter setting for metaheuristics in general. Their approach is based on a developed procedure called CALIBRA, however, this approach has the limitation of tuning only five parameters and the values obtained are not guaranteed to be the best. (Coy et al., 2001) have provided an approach based on a statistical design of the experiment and applied it to a vehicle routing problem. Their method, however, required a rough approximation based on human experience for the initialisation of the method itself. Research accomplished by Bartz-Beielstein and Markon (2004) proposes a method to determine relevant parameter setting based again on the statistical design of experiments and a tree based regression analysis. Birattari (2002) proposed a procedure that empirically evaluates a set of candidate configurations by discarding bad ones as soon as statistically sufficient evidence is gathered against them. It can be noticed from these publications on general parameter optimisation for metaheuristics that they are based on statistical analysis.

Other approaches have been carried out to optimise the parameters of ant systems. These approaches can be divided into three groups; the first group is that of proposed methods to find the best parameter settings for one or some of the parameters, the second group proposes experimental analysis to find the proper parameter set, while the third group tries to connect the problem instance with the optimal parameters.

Of the first group (Dorigo and Gambardella, 1997) presented a formula for the optimum number of ants based on the value of ρ and q_0. Watanabe and Matsui (2003) proposed an adaptive control mechanism of the parameter candidate sets based on the pheromone concentration for improving the ACO algorithms. Zecchin et al., (2005) developed parametric guidelines for the application of the ACO to the optimisation of a water distribution system. Hao et al., (2006) have chosen three parameters to optimise (β, ρ and q_0) and have developed a parameter study strategy based on PSO. A hybridised algorithm using GA and ACS-TSP was attempted by Pilat and White (2002) to solve the TSP faster but their attempt failed to bring results better than those of the original ACS (Dorigo and Gambardella, 1997). Pellegrini et al., (2006) employed the F-Race algorithm to find the four optimum parameters (m, ρ, α and q_0) for the Max-Min Ant Colony System (MMAS).

Of the second group Pilat and White (2002) proposed a Meta ACS using the GA as another layer wrapping the ACS to optimise its parameters. They had better results on their study which considered only three parameters (β, ρ, q_0). Socha (2003) proposed computational studies on some parameters. Also, Solnon (2002) made computational studies on some parameters of the ACS as a pre-processing step.

Of the third group Gaertner and Clark (2005) presented a design of an experiment based on an exhaustive search to find the optimum values of the three parameters (β, ρ and q_0) for a TSP instance. In doing so they tried to connect the TSP class with the optimum set of parameters. Also, Figlali et al. (2005) investigated the parameters of the ACS with the randomly generated job-shop scheduling problem.

2.3.6.6 ACO and Local Search

Most ACO algorithms are coupled with operators or local search techniques. The outcome in performance has been similar to that of the best heuristic approaches employed (Stützle and Hoos, 1997).

Local search techniques such as 2-Opt or 3-Opt work as improvement heuristics. When an ACO algorithm provides a feasible tour, local search techniques repeatedly perform operations (exchanges or moves) which reduce the tour length until no further improvement is possible (these operators will be explained in details in chapter 5 section 5.2.2).

2.3.6.7 Applications and optimisation problems

Ant colony optimisation algorithms have been applied to solve a range of combinatorial (Dorigo et al., 1999) and continuous (Dréo and Siarry, 2004) optimisation problems. ACO was first employed to tackle combinatorial problems like the TSP, scheduling problems e.g. the job-shop scheduling problem (Figlali et al. 2005), the vehicle routing problem (Farooq, 2008), the quadratic assignment problem (Stützle, 1997) and they were also employed as classifiers (Martens et al., 2007). The ACO was then employed for continuous optimisation by Bilchev (Bilchev and Parmee, 1995; Mathur et al., 2000).

2.3.7 Honeybees Inspiration: Behaviour, Algorithms and Application

Honeybees inspired algorithms are a branch of Swarm Intelligence algorithms which are motivated by the fascinating behaviour of honeybees. Their behaviour is studied in order to develop metaheuristic algorithms which can mimic the bees searching abilities. The algorithms are then employed to find solutions to real life problems. Four forms of honeybee behaviour have emerged in the literature, namely, the nesting site selection (Seeley and Visscher, 2003; Passino et al., 2008), the mating behaviour (Haddad et al., 2006; Sung, 2003), the honeybee teamwork strategy (Sadik et al., 2006) and the foraging behaviour (Seeley, 1996) and. These types of behaviour have been modelled to derive various Bees Algorithms with many applications.

2.3.7.1 The Nesting Site Selection

A swarm of honeybees choosing its future home is one of the most impressive examples known of an insect group functioning as an adaptive decision maker. In honeybee nest-site selection (Seeley and Visscher, 2003) when the size of the hive becomes too small for the honeybees to live in, a swarm of half the old colony with the mother queen flies a few meters from the hive and gathers in a tree or on a branch to form a cluster. Then only the scout bees from this swarm cluster begin to search for potential nest sites in all directions and at distances of up to several kilometres from the swarm.

Scouts assess the quality of sites based on cavity volume, entrance height, entrance area, and other attributes that are likely correlated with colony success. A dozen or more potential nest sites are initially advertised by the returning bees through a representative dance they perform. Eventually the bees advertise just one site which is not necessarily the one that was first advertised to the swarm.

During the first stages of this optimisation technique, each returning bee advertising a site is watched by other ‘‘unemployed’’ scouts which seek to observe dances. If they easily find a dancer they get recruited to a relatively high quality site. If they must wait too long to find a dancer, this would indicate that there are not many good nest-sites currently being assessed so they explore the environment for more sites. The number of recruits to each nest-site is in proportion to the number of dances for each site.

At each nest-site there is a quorum-sensing process, where once there are a certain number of bees at the site, the bees from that site ‘‘choose it’’ by returning to the cluster to prompt lift-off and then they guide the swarm to its new home (Beekman et al., 2006). There is significant time–pressure to complete the nest-site selection process as fast as possible since weather and energy losses pose significant threats to an exposed colony. However, enough time must be dedicated to ensure that many bees can conduct independent evaluations of the site and establish a quorum at a site that is likely to be the best site that the swarm has found. Hence, during nest-site selection the swarm strikes a balance between time minimisation and site quality choice maximisation. Agreement among the dancers appears and within an hour or so of the appearance the swarm lifts off. There is an increase of dancing just before liftoff. The analysis of the dancing records of individual scout bees confirmed that there is much turnover in the dancing bees over the course of a decision making process (Passino et al., 2008). Most bees that dance for a site cease doing so after a few hours, letting the next "generation" of dancers carry on the deliberations. Thus it became clear that a choice of a swarm of a future home is broadly distributed among the scout bees, and that this leaderless process of group decision-making consists of friendly competition among the different groups of dancers representing the different potential nest sites. The groups compete for additional dancers. Sooner or later, one group of dancers grows numerous and ultimately excludes its competitors. The site whose dancers prevail in this winners-take-all contest becomes the new home of the swarm.

2.3.7.2 Honeybees Mating Behaviour

Each normal honeybee colony consists of the queen, drones, workers, and broods. Queens represent the main reproductive individuals in some types of honeybees and specialise in laying eggs. Drones are the sires or fathers of the colony. They are haploid and act to amplify the genome of their mothers without alteration of their genetic composition except through mutation.

Therefore, drones are considered agents that propagate one of the gametes of their mother and function to enable females to act genetically as males. Workers specialise in brood care and sometimes lay eggs. Broods arise either from fertilised or unfertilised eggs. The former represent potential queens or workers, whereas the latter represent prospective drones.

The marriage process in honeybees was hard to observe as the queens mate during their mating flight far from the nest. A mating flight starts with a dance performed by the queens who then start a mating flight during which the drones follow the queens and mate with them in the air. In a typical mating-flight, each queen mates with seven to twenty drones. In each mating, sperm reaches the spermatheca and accumulates there to form the genetic pool of the colony. Each time a queen lays fertilised eggs, she retrieves at random a mixture of the sperms accumulated in the spermatheca to fertilise the egg.

Just as the queen bee is the only bee in a hive that breeds with the others, the best solution in the pool of solutions is selected to crossbreed with a random set of others. Thus the algorithm aims to retain the best solutions in the "gene pool," and achieve a better answer. This differs from the traditional approach, which selects both parents randomly from the whole pool (Abbass, 2001). This natural behaviour also led to the Queen-Bee Evolution method where in a generation the fittest individual crossbreeds with the other bees selected as parents by a selection procedure. This method when combined with a GA increases the exploitation of the GA. However, it also increases the probability that the GA will fall into premature convergence and results in a decrease in the performance of the GA. To decrease the probability of premature convergence and to reinforce the exploration of a GA, some individuals in Queen-bee evolution are strongly mutated (Sung, 2003; Azeem and Saad 2004).

2.3.7.3 Honeybee teamwork strategy

In Honeybees, the queen controls the nest and all the other bees provide various services to the queen. Honeybees move from flower to flower extracting nectar which they deliver back to the nest where it is employed to make honey. When a queen dies, a new queen is raised by feeding a normal worker bee with special food.

Abstract mapping has been done of similarities between this Honeybee behaviour and agent teamwork strategies, which are later employed in the design of teamwork architecture and elaborated using prototype case studies (Sadik et al., 2006).

2.3.7.4 Foraging Behaviour of the Honeybees

Honeybees can exploit a vast number of flower patches by extending their search over enormous fields surrounding the hive. They search for flower patches that provide plentiful amounts of nectar or pollen that are easy to collect with less energy usage (Frisch, 1976; Seeley, 1996).

The foraging process during the harvesting season begins in a colony by employing scout bees to search for adequate flower patches where nectar is plentiful, easy to extract, and rich in sugar content. Scout bees search randomly through their journey from one patch to another. Moreover, during the whole harvesting season, a colony continues its exploration, keeping a percentage of the whole population as scout bees (Seeley 1996).

When they return to the hive, those who have found a high-quality food source that is above a certain threshold (a combination of certain constituents, such as sugar percentage), they deposit their nectar or pollen that they have collected during the search process and then signal the position of their discovery to resting nestmates through a ritual known as the "waggle dance" on the dance floor (Frisch, 1976). The mysterious waggle dance is essential for colony communication and is performed in a particular area of the hive called the "dance floor", and communicates three basic pieces of information regarding the flower patch: the direction where it is located, its distance from the hive, and its quality rating (Frisch, 1976; Camazine et at., 2003). After the waggle dance, the dancer bee goes back to the flower patch, followed by other nestmates recruited from the hive. The number of recruited bees depends on the quality rating of the patch. Flower patches that contain rich and easily available nectar or pollen sources attract the largest number of foragers (Bonabeau, 1998; Seeley, 1996). The information guides the colony to send its bees to flower patches precisely, without any supervisory leader or blueprint. The knowledge of each individual of the outside environment is gleaned solely from the waggle dance. This dance gives the colony a chance to evaluate different patches simultaneously in addition to minimising the energy usage rate (Camazine et al., 2003). This allows the colony to gather food quickly and efficiently.

While harvesting the source, the bees monitor the food level. This information will be necessary when deciding on the next waggle dance when they return to the hive (Camazine et al., 2003). If the food source is still good enough and calls for more recruitment, then that patch will be advertised by making a waggle dance and recruiting more bees to that source.

Once a recruited forager returns to the hive, it will in turn waggle dance to direct other idle bees towards the food source. Thanks to this mechanism, the most profitable food sources attract the largest number of foragers (Tereshko and Lee, 2002), and thus the bee colony optimises the efficiency of the food collection process (i.e. the amount of food collected versus the cost of harvesting it).

During harvesting when the bees detect that there is no more nectar in a flower patch, bees will abandon it and interrupt the dances intended to attract other bees to that location.

A number of algorithms has been inspired by bee swarming behaviour and employed in discrete space, in the next section we will examine several:

1. BeeHive Algorithm

A model borrowing from the principles of bee communication is presented in (Wedde et al., 2004). The artificial bee agents are employed in packet switching networks to find suitable paths between nodes by updating the routing table. Two types of agents are employed – 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. Though the paper talks in terms of bees, it only loosely follows their natural behaviour.

The BeeHive algorithm (Wedde et al., 2004) was introduced and applied to routing problems in packet switching networks (Farooq, 2008) where agents called BeeAgents were employed to route packets among network nodes.

2. BeeAdHoc Algorithm

A new routing algorithm for energy efficient routing in mobile ad hoc networks was developed by (Wedde et al., 2005) based on the foraging behaviour of honeybees. The algorithm mainly utilises two types of agents, scouts and foragers, for doing routing in mobile ad hoc networks. The algorithm, BeeAdHoc, is a reactive source routing algorithm and it consumes less energy as compared to existing state-of-the art routing algorithms because in using the principals of foraging behaviour, it utilises less control packets to do routing. The results showed that the BeeAdHoc algorithm consumes significantly less energy as compared to state-of-the-art routing algorithms, without making any compromise on traditional performance metrics (packet delivery ratio, delay and throughput).

3. The Honey Bee Algorithm

A model generated by studying the allocation of bees to different flower patches to maximise the nectar intake is described in (Tovey, 2004). This was subsequently applied to distribute web applications at hosting centres.

4. The Virtual Bee Algorithm

In (Yang, 2005), the author describes a virtual bee algorithm where the objective function is transformed into virtual food. Unfortunately no information about how the transformation from objective function to food source or how agent communication is carried out is given in this work. Nor are there any comparative results to check the validity of the algorithm.

5. The ABC algorithm

For applications in the area of continuous function optimisation, Karaboga and Basturk (2008) proposed the ABC algorithm. Although the two algorithms were developed independently, there are strong analogies between the ABC and the BA. The two optimisation methods can be described using the same flowchart, and the site abandonment procedure is also employed in the ABC algorithm. Differently from the BA, the ABC uses the roulette wheel selection method (Karaboga and Basturk, 2008) to simulate the recruiting of foragers through the waggle dance. The main difference between the two algorithms is in the implementation of the local search procedure. The ABC generates foragers (parents) by a floating point crossover operation (Pham and Karaboga, 2000) between the dancer bee and a second bee randomly selected from the population. This operator calculates the components of the new forager as a weighted average of the components of the parents. The weight of each component of the parents is randomly determined. Since the second bee is randomly selected from the whole population, the crossover operation may generate a forager bee which is relatively far from the dancer bee. In particular, the forager bee may be placed outside the fitness peak that it is meant to exploit. For this reason, the effectiveness of the exploitative search may be reduced, and the extent of the neighbourhood search is more difficult to control.

6. Bee Colony Optimisation

Another implementation of bee behaviour was presented by (Teodorovic et al., 2006) to solve transportation problems and was called Bee Colony Optimisation. This algorithm uses a constructive approach that is similar to ACO.

2.4 THE BEES ALGORITHM AND ITS SUCCESSORS

The Bees Algorithm is a population based search algorithm that imitates the food foraging behaviour of honeybees to find the optimal solution for both continuous and combinatorial problem.

2.4.1 Standard Bees Algorithm

The Bees Algorithm balances between the global and the local search. The BA randomly explores the solution space looking for areas of potential optimality(s). Then the Bees Algorithm exploits these areas by conducting a local search, until either a satisfactory solution is found, or a pre-defined number of iterations has been reached. More detailed explanation of the Bees Algorithm is discussed in the next chapter.

2.4.2 Improvements and Applications

Various versions of the Bees Algorithm were successfully developed to solve different engineering problems more efficiently. Their application was also extended to a vast number of new continuous and combinatorial optimisation problems.

The Bees Algorithm was first applied to solve continuous function optimisation (Pham et al., 2006b). The Bees Algorithm was tested on a range of well-known benchmark function optimisation problems of different degrees of complexity and the experimental results proved the reliability of the Bees Algorithm (Pham and Castellani, 2009).

The Bees Algorithm was also applied to train different types of neural network such as the training of the Learning Vector Quantisation networks (Pham et al., 2006a), the training of Multi-Layered Perceptrons neural network (Pham et al., 2006c; Pham et al., 2006e), the training of the Radial Basis Functions network (Pham et al., 2006d), the training of the Spiking Neural Networks (Pham and Sahran, 2006) and the training of the Support Vector Machine (Pham et al., 2007a). These trainings were for recognising patterns in control charts employed for identifying wood defects. In general, results showed the testing and training accuracies using the Bees Algorithm as a classifier were either higher than or very close to those accuracies produced by other classifiers.

Pareto multi-objective optimisation was another extension of the Bees Algorithm applied to the welded beam design problem. The objective of the design is to minimise the cost of fabrication while finding a feasible combination of weld thickness, weld length, beam thickness and beam width. The Bees Algorithm produced better results (less cost) than almost all the examined algorithms (Ghanbarzadeh, 2007).

The Bees Algorithm was also applied to environmental/economic power dispatch problems with weighted-sum multi-objective optimisation (Lee and Haj Darwish, 2008) and with Pareto optimality (Pham et al., 2008a).

A new formulation of the Bees Algorithm was proposed for solving a chemical engineering process as a dynamic optimisation problem. It includes new search operators, and a new selection procedure that enhances the survival probability of newly formed individuals. The proposed algorithm was tested on six benchmark dynamic optimisation problems. For all the problems, the Bees Algorithm found very satisfactory optima and the results proved the high reliability of the proposed technique (Pham et al., 2008b).

In a biological application involving protein structures, the Bees Algorithm was adapted to search the protein conformational search space to find the lowest free energy conformation (Bahamish et al., 2008). Proteins perform many biological functions in the human body. The structure of the protein determines its function. In order to predict the protein structure computationally, the protein must be properly represented. The algorithm was able to find the lowest free energy conformation.

One of the first implementations of the Bees Algorithm in discrete space for combinatorial problems was in designing cellular manufacturing systems where the Bees Algorithm was employed for solving the cell formation problem (Pham et al., 2007b). Experimental results indicated that the Bees Algorithm is very effective for large-scale problems.

Another application of the Bees Algorithm is the scheduling of jobs with a common due date for a machine to minimise the penalties associated with early or late completion (Pham et al., 2007c). Results proved it to be more stable and robust than other existing optimisation techniques such as GA, PSO and TS.

The Bees Algorithm was successfully applied for PCB assembly planning (Pham et al., 2007d). The computational experiments showed that the Bees Algorithm gives a significant reduction in assembly time compared to the results obtained with the GA and EP on a benchmark assembly task. The Bees Algorithm was also copped with TRIZ-inspired operators for the same application (Ang et. al, 2009) where a shorter time was obtained as compared to previous work.

The Bees Algorithm was also applied to solve the Scholar Timetabling Problem. Experimental results showed promising results (Lara et al., 2008).

Preliminary design is another application of the Bees Algorithm (Pham et al., 2007e). The algorithm has been also employed to design mechanical components (Pham et al., 2008c; Pham et al., 2009).

The algorithm was also employed to obtain the optimal sink path for large-scale sensor networks (Saad et al., 2008). Another usage of the Bees Algorithm was in robotics. It was employed for learning the inverse kinematics of a robot manipulator (Pham et al., 2008e).

The Bees Algorithm was also applied to clustering problems to improve the results of the K-means (Pham et al., 2007f) and the C-means (Pham et al., 2008f) algorithms

Interpolation and extrapolation operators were introduced to unselected bees in the Bees Algorithm by (Ghanbarzadeh, 2007) to improve these bees by mating them with the selected ones. Also, two new methods were proposed for the Bees Algorithm namely the ‘shrinking’ method for neighbourhood size and the idea of ‘abandon sites’ which is employed when stuck in a local optimum or when no new information is found.

In another arrangement, an adaptive neighbourhood search and random particles were added to the global search by proposing a hybrid PSO-Bees Algorithm to solve the problem of premature convergence in the basic PSO algorithm (Pham and Sholedolu, 2008).

2.5 SUMMARY

Various optimisation techniques and algorithms were briefly introduced in this chapter alongside the terms and definitions often employed in the study of optimisation. The natural inspiration of these algorithms and their application to the field of engineering were discussed. Applications of these algorithms to different type of problems were also examined.

CHAPTER 3

THE META BEES ALGORITHM FOR SOLVING CONTINUOUS OPTIMISATION PROBLEMS

This chapter describes the main characteristics of the standard Bees Algorithm and a study to explore the parameters of the algorithm to help understand the methods by which its performance is improved, such as those employed to avoid premature convergence. The chapter presents a new method of tuning the Bees Algorithm called Meta Bees Algorithm (a Bees Algorithm within a Bees Algorithm). The tuned Bees Algorithm has been applied to a range of function optimisation problems. The results obtained have been compared against those produced by other optimisation algorithms.

3.1 A STUDY OF THE STANDARD BEES ALGORITHM PARAMETERS

As mentioned in the previous chapter, the Bees Algorithm takes its inspiration from the food foraging behaviour of honeybees to search for the best solution to a given optimisation problem. As shown in Figure 3.1, the algorithm randomly samples the solution space looking for areas of high performance. Throughout the search it performs an exploitative neighbourhood search combined with a random explorative search. In order to study these two main searches performed by the Bees Algorithm, a set of experiments was run where the Bees Algorithm was applied to several function optimisation problems. The effects of the Bees Algorithm parameters are then discussed.

3.1.1 Random Initialisation

The bee population size is fixed to *n* scout bees. These bees randomly sample the solution space with uniform probability across the space. Each scout bee evaluates the visited site (i.e. solution) via the fitness function.

The population size *n* is one of the key parameters in the Bees Algorithm. The effect of changing the population size on the mean number of iterations to arrive at the correct answer is shown in Figure 3.2. The variation of population size against the number of evaluated points to arrive at the correct solution is shown in Figure 3.3. Also shown in Figure 3.4 is a graph demonstrating the performance of the algorithm with increasing population size.

Figure 3.2 shows that the number of iterations required to obtain the solution reduces with increasing population size.

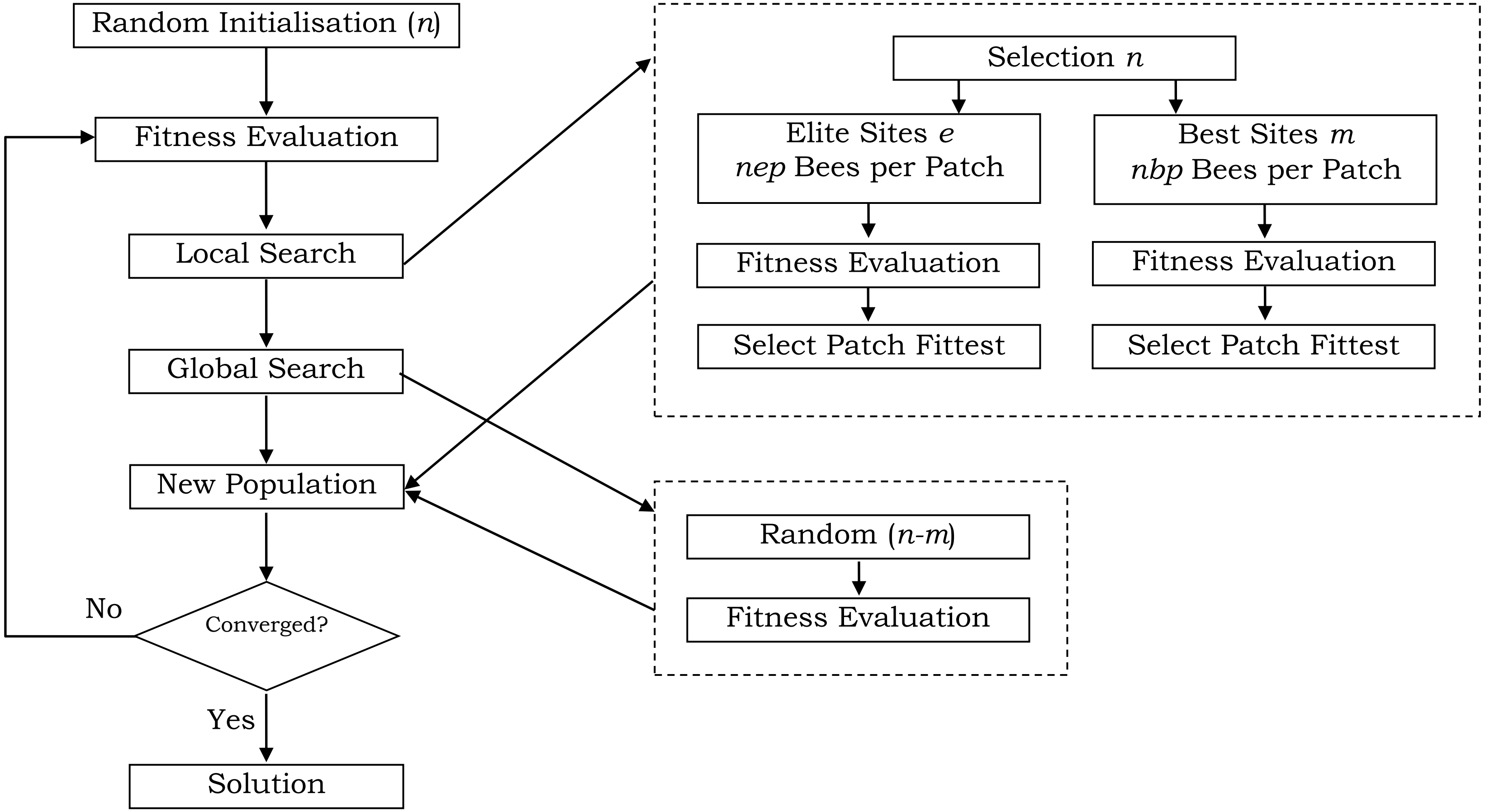


Figure 3.1 Flowchart of the Standard Bees Algorithm

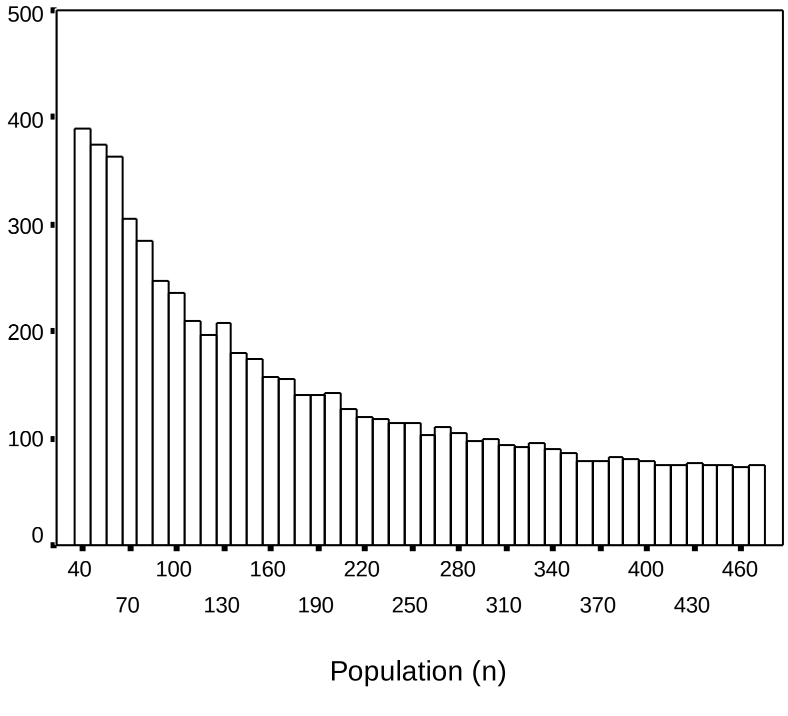


Figure 3.2 Mean Iteration versus Population Size

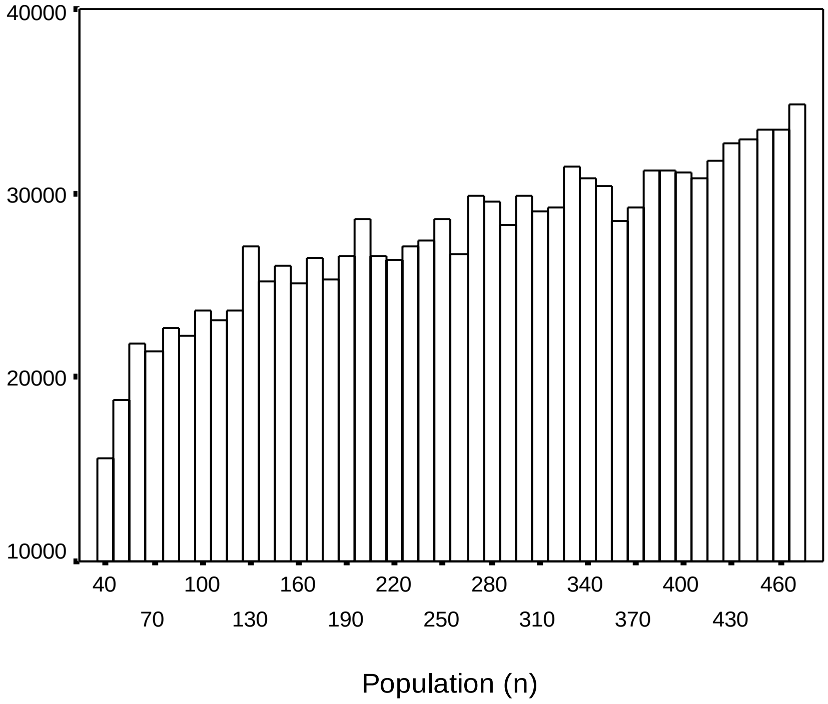


Figure 3.3 Mean Number of Function Evaluations versus Population Size

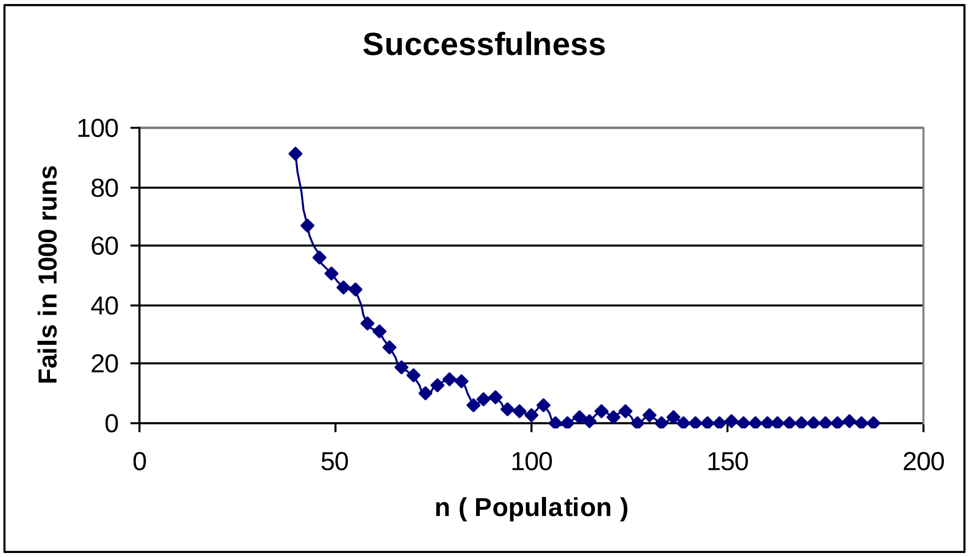


Figure 3.4 Performance for Different Population Sizes

Figure 3.3 shows how an increase in population size leads to an increase in the number of function evaluations, which is predictable. Here, the mean generated points is defined as the mean of the number of times the objective function was called.

To achieve higher algorithm reliability, a minimum size of population is required as shown in Figure 3.4 (where Fails represents a case where the optimum solution of the optimisation problem could not be reached). In order to arrive at the solution with fewer iterations, the population should be larger than the population used. To obtain a reasonable number of function evaluations, the population has to be as small as possible. Within these three constituent parts (the algorithm reliability, the number of iterations and the number of function evaluations) a range needs to be set to choose a proper population size.

3.1.2 Neighbourhood / Local Search

As in all the evolutionary algorithms, the neighbourhood search is an essential concept of the Bees Algorithm. After ranking the sampled solutions and locating the most promising ones (i.e. the highest ranking locations), other bees are recruited to search the fitness landscape in the neighbourhood of these solutions.

The neighbourhood search is based on a random distribution of bees in a predefined neighbourhood range. For each selected site, bees are randomly distributed to find a better solution. As shown in Figure 3.5, only the best bee is chosen to advertise its source after which the centre of the neighbourhood field is shifted to the position of the best bee (i.e. from A to B).

In undertaking the study of this form of local search, three issues have been taken into account; the number of recruited bees in the neighbourhood range, the width of range and the method of site selection.

(A) The Number of Recruited Bees

The number of recruited bees around selected sites should be defined properly. When the number is increased, the number of function evaluations will also be increased and vice versa, when the number decreases, the chance of finding a good solution decreases.

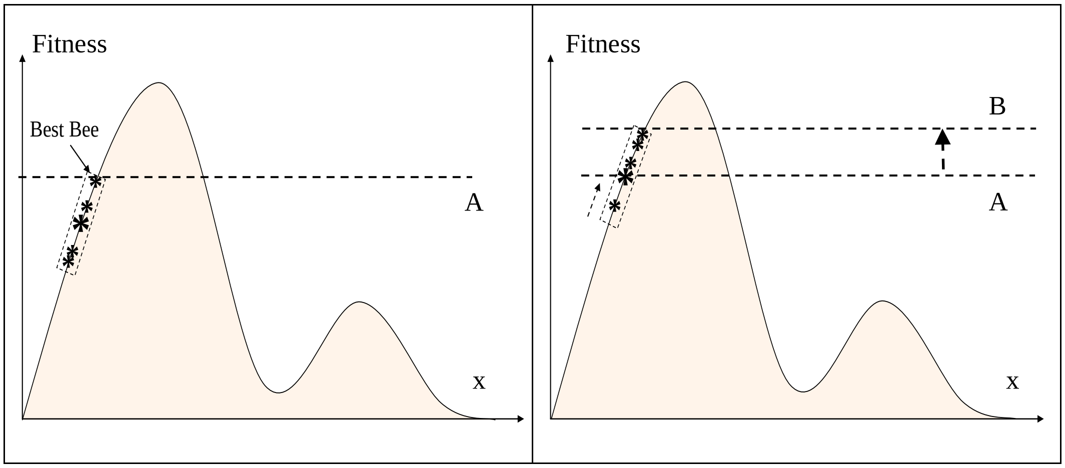


Figure 3.5 Graphical Explanation of the Neighbourhood Search

(B) The Patch Size (Neighbourhood Range)

When the neighbourhood range can be arranged adequately, the number of recruited bees will depend on the complexity of a solution space. The neighbourhood range is a variable which needs to be tuned for different types of problem spaces. Three different strategies have been applied to improve the efficiency and robustness of the Bees Algorithm. These are (1) a fixed neighbourhood region width strategy, (2) a region changing according to iteration strategy and (3) hybrid strategies combining the two previous strategies where, for instance, the first strategy can be employed up to the 50th iteration and then the second strategy employed up to the stopping criteria.

With the first strategy, a neighbourhood width for all selected sites was fixed to a certain range that is sufficient to deal with the complexity of the problem space. In this strategy, beginning from the first iteration, all the bees harvest on the same size fields which are defined as (*ngh*) in Equation (3.1).

$$\text{Neighbourhood\_Range} = ngh \qquad \text{Equation (3.1)}$$

However, in the second strategy, the region changes proportional to the number of iterations. All sites have the same range value and this range will be narrowed down depending on the iteration as shown in Equation (3.2). This strategy has been established to increase the accuracy of the solution.

$$\text{Neighbourhood\_Range} = (ngh \,/\, \text{iteration}) \qquad \text{Equation (3.2)}$$

Finally, the third strategy was implemented as a combination of the first two strategies to improve their efficiency.

These neighbourhood range strategies have been tested using the De Jong function benchmark presented in Table 3.1. These tests were run independently 10 times and the mean of these 10 runs is presented in Figure 3.6 with the number of iterations being set to 1000. The parameters set for this test were as follows:

The population is set to 15, the selected sites are 5, the elite sites 2, bees around elite points numbered 4, bees around selected points numbered 2. A neighbourhood spread of 0.01 is defined as an initial range for all strategies. For the third, mixed, strategy the first 20 iterations of the algorithm employed the first strategy (fixed ranges) and thereafter the second strategy of narrowing down the ranges was employed. It is clear from the results in Figure 3.6 that the first strategy reaches its minimum value before any of the other strategies.

However, the second and third strategies gave similar results for this relatively simple problem space. Thus, it does not make sense to employ the more complex second and third methods as the first strategy can be said to be both a simple and efficient method for neighbourhood searching.

Table 3.1 Test Functions (Mathur et al., 2000)

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	Martin and Gaddy	[0, 10]	$\min F = (x_1 - x_2)^2 + \frac{(x_1 + x_2 - 10)^2}{3}$	X(5,5)
				F=0

3	Rosenbrock (2D)	[-1.2, 1.2]	$\min F = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	X(1,1)
				F=0

4	Rosenbrock (2D)	[-10, 10]	$\min F = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	X(1,1)
				F=0

5	Griewank	[-600, 600]	$\min F = \frac{1}{4000} \cdot \left(\sum_{i=1}^{50} (x_i - 100)^2 \right) - \left(\prod_{i=1}^{50} \cos\left(\frac{x_i - 100}{\sqrt{i}} \right) \right) + 1$	X($\vec{1}00$)
				F=0

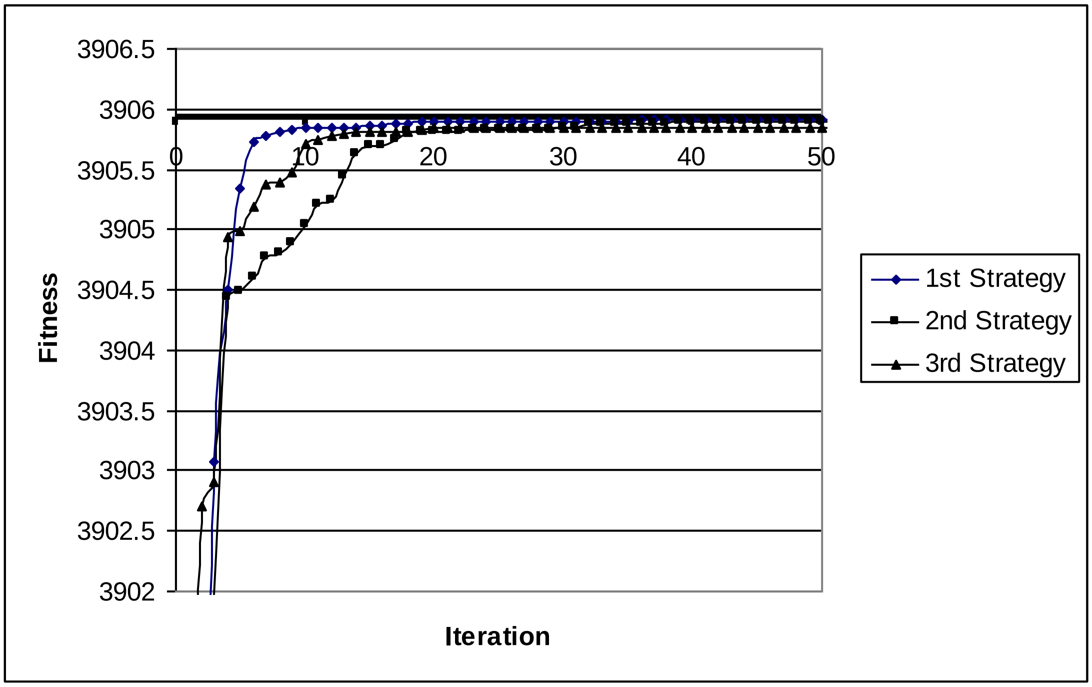


Figure 3.6 Comparison of Different Neighbourhood Strategies for the De Jong Function

(C) The Recruitment Strategy (Site Selection)

Two different techniques were implemented: Probabilistic Selection and Best Site Selection.

With the Probabilistic Selection technique, the roulette wheel method is employed and sites with better fitness have more chance of being selected. However, with the Best Selection technique (greedy selection), the best sites according to fitness will be selected. Different combinations of the selection methods, ranging from pure Probabilistic Selection (q=0) to pure Best Selection (q=1), have been investigated. The mean number of iterations required to reach the answer and the success of each combination are shown in Figures 3.7 and 3.8.

From the experimental results, the Best Site selection technique demonstrated higher success. It is simpler to implement as is does not involve the use of a roulette wheel which causes the algorithm to take longer and makes it more complicated, and thus the Best Site selection has been recommended for use.

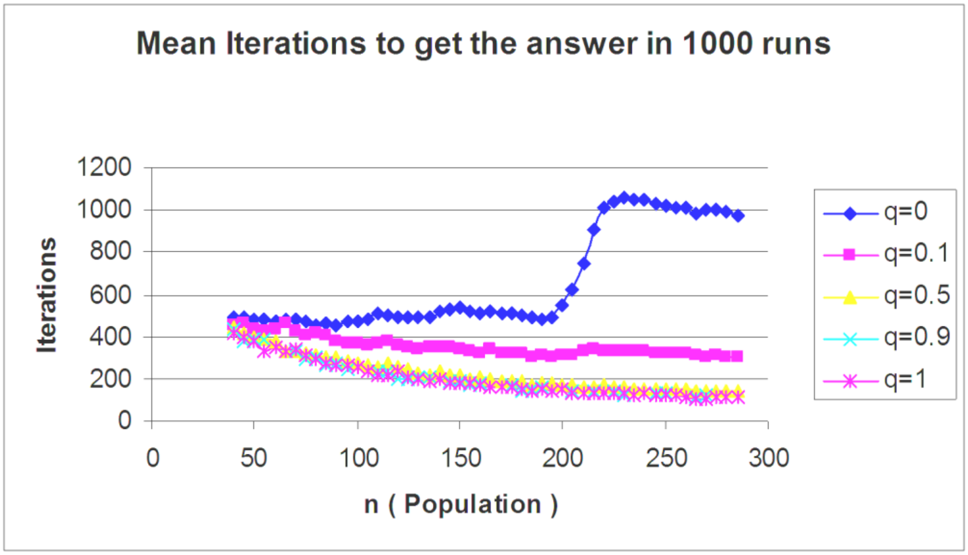


Figure 3.7 Mean Iterations required for Different Combinations of methods

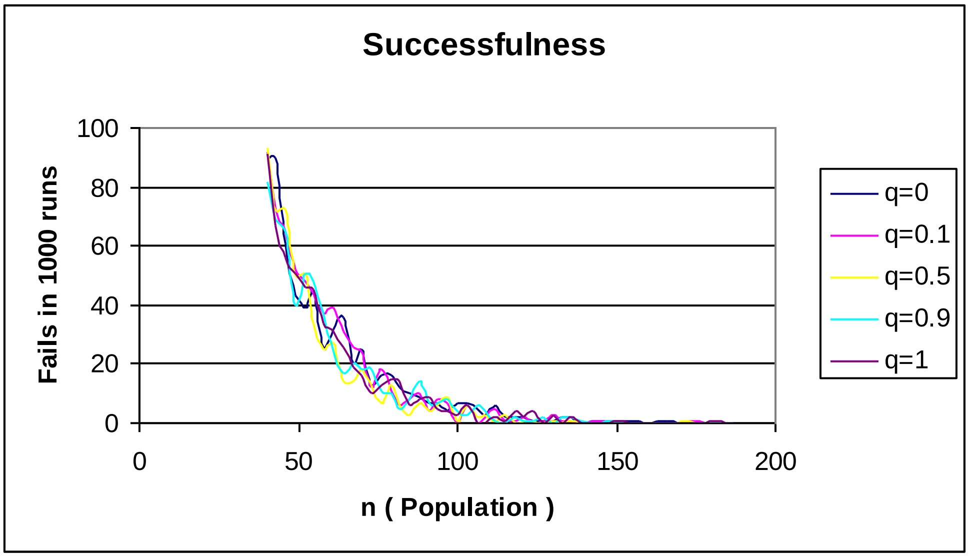


Figure 3.8 Performance of Different Combinations of the Selection Method

3.2 THE BEES ALGORITHM PARAMETER TUNING

The Bees Algorithm, like all other metaheuristic search algorithms including Tabu Search (TS), Simulated Annealing (SA) and ACO, invariably requires a set of parameters in order to solve complex optimisation problems. These parameters directly impact on the performance of the solver and as such, the researcher will often hand-tune parameter values before the application of the Metaheuristic or use a standard set of values that have been found to be traditionally well-suited by other researchers.

As shown in the first section of this chapter, the search strategy of the Bees Algorithm combines global random exploration with local neighbourhood sampling. The explorative search (scout bees) and exploitative search (recruited foragers) are clearly differentiated and they can be independently varied through a set of learning parameters. This clear decoupling between exploration and exploitation facilitates the tuning of the algorithm.

In spite of the fact that most of the work that has been carried out in this field states the need for a mechanism to tune the Bees Algorithm parameters, comparatively very little research has been carried out into the analysis of the parameter values or the way they can be automatically derived or tuned. The optimisation of the Bees Algorithm has been carried out, in the past, according to experimental trial and error and the values of its parameters were decided empirically. Once the learning parameters were manually optimised by conducting a number of trials, they were fixed and kept unchanged for all the optimisation problems. A trial to understand the effect of the parameters of the Bees Algorithm on its performance and the speed of convergence had been carried out (Pham et al., 2005). The experiments were conducted to test different parameters of the algorithm and the effects of changing one parameter while keeping other parameters fixed. Researchers (Pham et al., 2006b) have indicated that one of the drawbacks of the original Bees Algorithm is the number of tuneable parameters used and that further work should target the reduction of the number of learning parameters. There was a trial conducted by (Pham et al., 2007d) to drop these numbers which concluded that even though the performance of the Bees Algorithm is fairly robust to reasonable variations of the learning parameters, a smaller parameter set would ease the optimisation of the algorithm performance.

No research has been conducted into either the statistical analysis of the Bees Algorithm parameter values or the way they can be automatically derived or tuned by metaheuristic algorithms themselves. However, it is possible to run the Bees Algorithm on top of another Bees Algorithm during the parameter search process and find the optimal settings, although this may incur a computational overhead.

The proposed combined Bees Algorithm, called Meta Bees Algorithm, is used to evolve suitable parameter values using its own optimisation process while solving complex problems.

3.2.1 The Meta Bees Algorithm

In this chapter a technique employed by the Bees Algorithm allows it to self-adapt its own parameters to minimise the sensitivity of these parameters by finding an area where the effects of these parameters on the algorithm are less. The approach adopted uses a standard mechanism of the Bees Algorithm to modify and determine the appropriate parameter values while the problem is being solved. Therefore it is conceptually simple to integrate this approach into the standard Bees Algorithm.

For this work, the Bees Algorithm was ‘wrapped around’ a second Bees Algorithm (the Wrapper). Figure 3.9 shows the pseudocode of the Meta Bees Algorithm and Figure 3.10 demonstrates the flow of information between the two Bees Algorithms. It can be seen that each bee of the Wrapper Bees Algorithm represents a set of Bees Algorithm parameters and, again, the fitness values are the total number of function evaluations required by the Bees Algorithm to find the optimum solution for the function optimisation. The Bees Algorithm uses the same mechanisms for generating solutions to evolve appropriate values for its parameters.

3.3 COMPUTATION EXPERIMENTS

3.3.1 The Function Optimisation Problem

The performance of the Meta Bees Algorithm was evaluated on a set of five continuous function minimisation benchmarks (Mathur et al., 2000). Using the proposed method for each problem, the results obtained were compared with the results given by the standard Bees Algorithm an with the other optimisation algorithms. The function minimisation problems represent a varied set of learning scenarios that were chosen from amongst widely used benchmarks in the literature (Mathur et al., 2000). Table 3.1 shows the equations of the five continuous function minimisation benchmarks. For each function, the equation is given together with the range of the variables and the global minimum. The *Martin and Gaddy* benchmark is a fairly simple unimodal function. The *Rosenbrock* benchmark is unimodal, the minimum lies at the bottom of a long, narrow, parabolic shaped valley. Finding the valley is trivial, however locating the minimum is difficult. The *Griewank* function has an overall unimodal behaviour, with a rough multi-modal surface created by a cosinusoidal “noise” component.

```
1- Initialise the Bees Algorithm Wrapped population with random parameter values from the Wrapper Bees Algorithm
2- Evaluate fitness of the population of the Wrapped Bees Algorithm.
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 fitness.
8- Select the fittest bee from each patch.
9- Assign remaining bees to search randomly and evaluate their fitness.
10- End While.
```

Figure 3.9 Pseudocode of the Meta Bees Algorithm

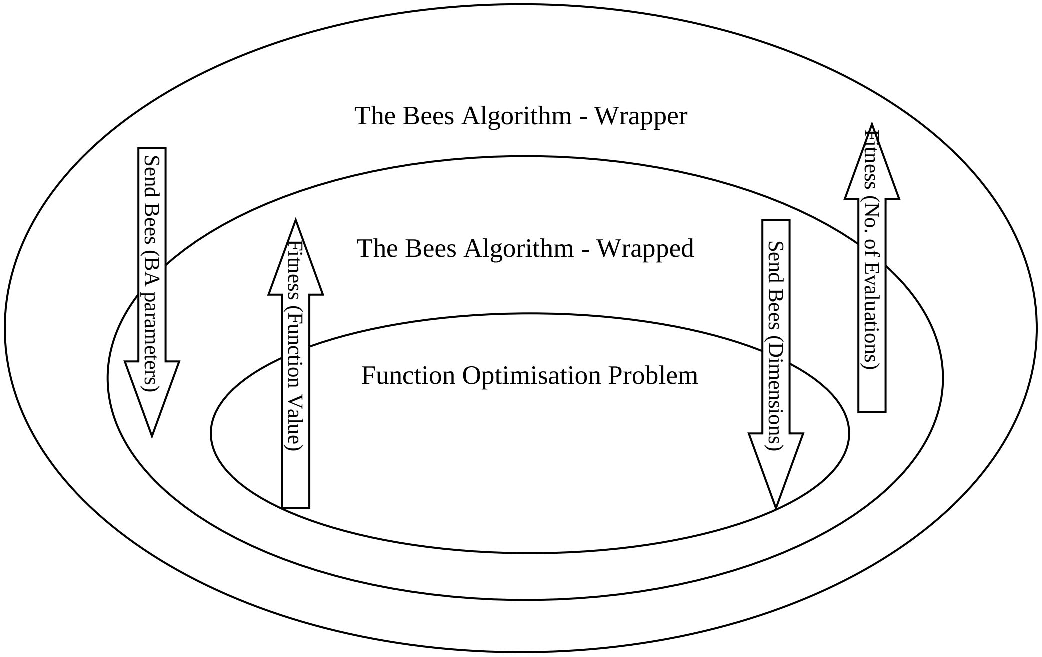

The Bees Algorithm - Wrapper
The Bees Algorithm - Wrapped
Function Optimisation Problem
Send Bees (BA parameters)
Fitness (Function Value)
Send Bees (Dimensions)
Fitness (No. of Evaluations)

Figure 3.10 A Bees Algorithm Wraps another Bees Algorithm

3.3.2 The Meta Bees Algorithm Implementation

The Bees Algorithm is characterised by a number of core parameters which are: the number of scout bees, *n*, the number of high-quality sites that are selected for neighbourhood search, *m*, the number of elite (top-rated) sites amongst the best *m* sites, *e*, the number of bees recruited for a neighbourhood search around the *e* elite sites, *nep*, the number of bees recruited for a neighbourhood search around the remaining (*m*-*e*) sites, *nsp*, the initial size of each flower patch, *ngh*, and the stopping criterion.

In addition to the above seven main parameters, two extra parameters have been introduced by Ghanbarzadeh (2007) in his thesis; the neighbourhood shrinking parameter and the site abandonment parameter. However, these two extra parameters have not been included in this study as the aim of the research was to show how well the standard Bees Algorithm can perform without the need to introduce extra parameters.

At each iteration, the standard Bees Algorithm is augmented by allowing each bee to select a value for each parameter before commencing the selection of the solution components. Thus each bee maintains its own parameter values and in turn uses these to adapt the parameter values. Selection of a parameter value is based exclusively on the fitness of the solution i.e. if an improvement in the fitness is found by a particular value of a set of parameters, the search exploits around this value to get a better result and so on.

As shown in Figures 3.10 and 3.11, the Meta Bees Algorithm consists of two Bees Algorithms namely the Wrapper Bees Algorithm and the Wrapped Bees Algorithm. The parameters values of the Wrapper Bees Algorithm have been setup empirically by using a pilot test and these are; *n1* = 10, *m1* = 3, *e1* = 1, *nep1* = 5, *nsp1* = 3, *ngh1* = 0.1 and the stopping criterion is 10000 iterations or 50000 numbers of evaluations.

Each parameter of the Wrapped Bees Algorithm of the Meta Bees Algorithm is given a suitable range in which its value can lay (Table 3.2). The initial value of each parameter is randomly chosen within the range.

Every time a value needs to be assigned to a parameter, a point will be randomly generated within the range given in the table (3.2) with the following two constrains (Equations 3.3 and 3.4)

$$m \leq n \qquad \text{Equation (3.3)}$$

$$e \leq m \qquad \text{Equation (3.4)}$$

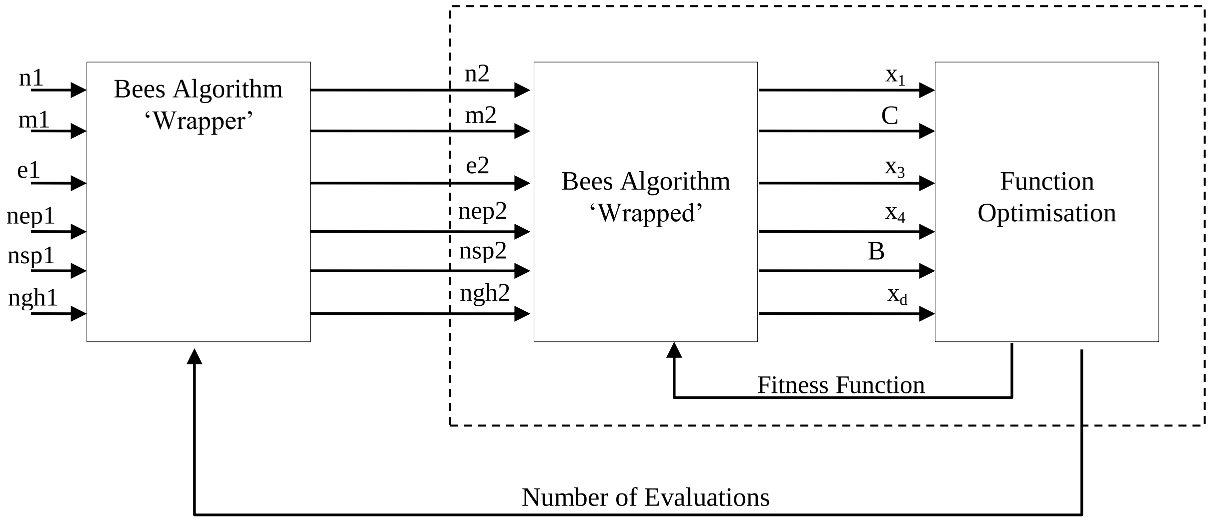


Figure 3.11 Illustration of the Meta Bees Algorithm

Table 3.2 Range of Values Available to the wrapped Bees Algorithm

Bees Algorithm Parameters	**Symbol**	**Value**	
		Minimum	**Maximum**
Number of scout bees	*n*	3	20
Number of best selected sites	*m*	3	10
Number of elite sites amongst the best m sites	*e*	1	10
Number of bees recruited for neighbourhood search around the e elite sites	*nep*	1	10
Number of bees recruited for best m sites	*nsp*	1	5
The initial size of each flower patch	*ngh*	0.01	0.2

3.3.3 Experimental Results

The experiments were performed using the Meta Bees Algorithm to evolve its own parameter values. It was run 100 times for each parameter setting on each benchmark problem. For each of the 100 trials, the Wrapped Bees Algorithm was run until it located a solution □ of fitness *F*(□) < 0.001 (Equation 3.5), or 50000 learning cycles had elapsed. The number of evaluations was recorded. The final accuracy result (E) was found to be:

$$E = \begin{cases} 0 \Leftarrow F(x_f) \leq 0.001 \\ F(x_f) \Leftarrow else \end{cases} \qquad \text{Equation (3.5)}$$

Where x_f is the final solution generated by the algorithm.

The computing platform used to perform the experiments was a 2.00GHz Intel(R) Core(TM) 2 Dual CPU PC with 1.99 GB of RAM. The experimental programs were coded in the C language and compiled with Microsoft Visual C++. Each problem instance was run across 100 random seeds.

3.3.4 Comparison between the Standard Bees Algorithm and the Meta Bees Algorithm

Table 3.3 shows the empirically derived Bees Algorithm parameter values used with the different test functions (Pham et al., 2006b).

Table 3.4 shows the results of running the Meta Bees Algorithm to optimise the parameter values employed in the Wrapped Bees Algorithm while solving the minimum function optimisation problem.

The characteristic values of each parameter are summarised as follows:

- *n*: The values were generally between 3 and 7, with 4 being most common.
- *m*: The value was the same (3) for all the optimisation functions.
- *e*: The value was the same (1) for all the optimisation functions.
- *nsp*: The value was the same (1) for all the optimisation functions.
- *nep*: The range of values was between 7 and 8, with 8 being the more common.
- *ngh*: Generally the range of values was between 0.02 and 0.06 with 0.02 being the most common.

Table 3.3 Standard Bees Algorithm Parameter Settings

Function No.		*n*	*m*	*e*	*nsp*	*nep*	*ngh* *(initial)*
1	De Jong	10	3	1	2	4	0.1
2	Martin and Gaddy	20	3	1	1	10	0.5
3	Rosenbrock (2D)	10	3	1	2	4	0.1
4	Rosenbrock (2D)	6	3	1	1	4	0.5
5	Griewank	50	5	2	10	20	5

Table 3.4 Meta Bees Algorithm Parameter settings

Function		*n*	*m*	*e*	*nsp*	*nep*	*ngh*
1	De Jong	4	3	1	1	8	0.03
2	Martin and Gaddy	4	3	1	1	8	0.02
3	Rosenbrock (2D)	7	3	1	1	7	0.04
4	Rosenbrock (2D)	4	3	1	1	8	0.02
5	Griewank	3	3	1	1	8	0.06
Most common value		**4**	**3**	**1**	**1**	**8**	**0.02**

Table 3.5 shows a comparison of the results obtained with the Meta Bees Algorithm and those obtained using the standard Bees Algorithm (Ghanbarzadeh, 2007), 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 (ACS) (Mathur et al., 2000).

The average number of evolutions i.e. the numbers of points visited is shown for 100 independent runs in Table 3.5. Inspection of the runtimes (number of evaluations) for both algorithms, the Meta Bees Algorithm and standard Bees Algorithm, reveal that overall, the Meta Bees Algorithm produced more stable values for the Bees Algorithm parameters with a smaller number of evolutions for most of the minimisation functions except the second function (Martin and Gaddy).

A T-Test and Random Distribution test were performed to evaluate the results obtained using the standard Bees Algorithm and Meta Bees Algorithm to see if their means are statistically different from each other. These were to ensure rigorous statements could be made regarding each set of results.

Table 3.5 Experimental Results

Funct. no.	SIMPSA		NE-SIMPSA		GA		ACS		Standard Bees Algorithm (*)		Meta Bees Algorithm	
	succ %	Mean no. of evaluations	succ %	Mean no. of evaluations	succ %	Mean no. of evaluations	succ %	Mean no. of evaluations	succ %	Mean no. of evaluations	succ %	Mean no. of evaluations
1	***	***	***	***	100	10160	100	6000	100	868	100	**683**
2	***	***	***	***	100	2844	100	1688	100	**526**	100	608
3	100	10780	100	4508	100	10212	100	6842	100	631	100	**571**
4	100	12500	100	5007	***	***	100	7505	100	2306	100	**1471**
5	***	***	***	***	100	200000	100	50000	100	20998	100	**16920**

*** Data not available

* (Pham et al., 2006b)

3.3.5 Statistical Analysis of the T-Test

An optimisation algorithm is more robust and stable when the variance of a performance criterion over a number of simulation runs is small (Engelbrecht, 2005). Engelbrecht showed the robustness of a swarm to be in the range:

Equation (2.1)

$$\text{Robustness}(S(t)) = [\theta - \sigma_\theta, \theta + \sigma_\theta]$$

(Engelbrecht, 2005)

Where θ is the average of the performance criterion over a number of simulation runs, and σ_θ is the variance in the performance criterion. The smaller the value of σ_θ the smaller the range performance values unto which the simulations converge – the more stable the swarm.

To check the statistical significance of the result by the Bees Algorithm, a T-Test is performed. The T-Test checks the relationship between two variables, in this case two different sets of parameters of the same algorithm and it tries to answer two questions:

1. What is the probability that a relationship exists?

2. If it does, how strong is the relationship?

In other words, tests for statistical significance are employed to address the question: what is the probability that the relationship between two variables is really just an occurrence of chance?

T-Tests are often employed in several different types of statistical tests:

- to test whether there are differences between two groups on the same variable based on the mean (average) value of that variable for each group;
- to test whether a mean (average) value of a group is greater or less than some standard;
- to test whether the same group has different mean (average) scores on different variables;

The test is employed for comparing the means of two samples even if they have different numbers of replicates. The test compares the actual difference between two means in relation to the variation in the data (expressed as the standard deviation of the difference between the means). A null hypothesis or an expectation to test against is required. In this case, for the Bees Algorithm and the Meta Bees Algorithm, the null hypothesis is that there is no difference in the performance of the two algorithms. The T-Test will help to decide if the data is consistent with this or departs significantly from this expectation. The T-Test assesses whether the means of two groups are statistically different from each other by providing an alpha (α) parameter. The parameter has three ranges: Where

- $\alpha < 0.05$, there is a significant difference in the group means.
- $\alpha < 0.01$, there is a more significant difference in the group means.
- $\alpha < 0.001$, there is a most significant difference in the group means.

The formula for the T-Test is a ratio. The numerator of the ratio is the difference between the two means or averages. The denominator is a measure of the variability or dispersion of the scores.

3.3.5.1 T-Test Results

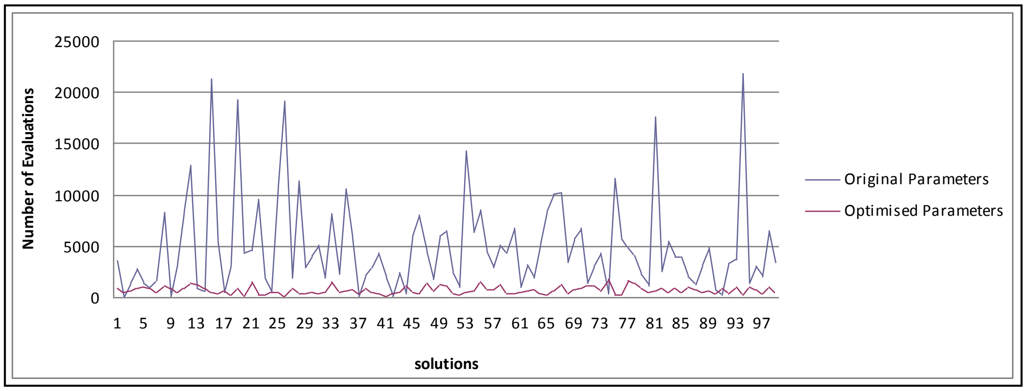
The first test applied to the obtained results was the T-Test (Schneider and Kirkpatrick, 2006). The T-Test was applied to all pairs of solutions and the difference in mean values was found to be significant at the 99% confidence level. That is, the Meta Bees Algorithm requires significantly fewer evaluations to solve the benchmarking functions than the Bees Algorithm, as shown in Figure 3.12 (a-e).

To demonstrate this significance graphically, the number of evaluations for each independent run for both the Bees Algorithm and Meta Bees Algorithm that were required to optimise the five functions studied in this chapter are shown in Figure 3.12 (a-e) (in figure (e), the line representing the original parameters is on the X axis and cannot be shown). It can be seen that on occasion the Bees Algorithm required fewer evaluations but that on average, over 100 runs, the Meta Bees Algorithm required fewer evaluations. The T-Test was conducted on the Meta Bees Algorithm and the original Bees Algorithm. Table 3.6 shows the alpha values of each test function after applying the T-Test to these functions.

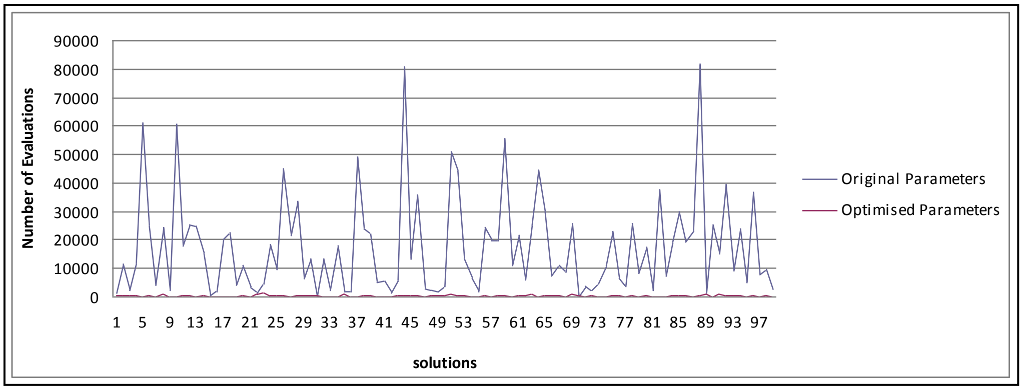
These values indicate that the results obtained by both the Meta Bees Algorithm and the original Bees Algorithm are most significantly different with a confidence level above 99%.

Table 3.6. The Values of Alpha of each Test Function

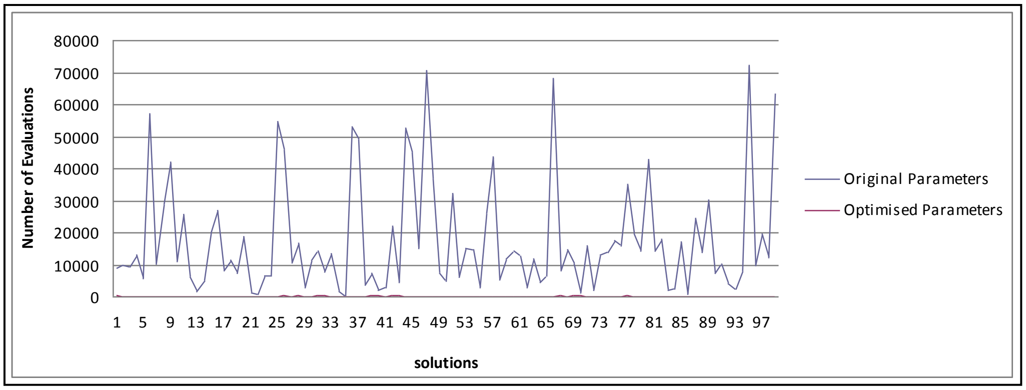
Funct. no.	1	2	3	4	5
Alpha	7.135×10^{-17}	1.148×10^{-19}	$2.646\times^{-19}$	5.735×10^{-16}	1.235×10^{-15}



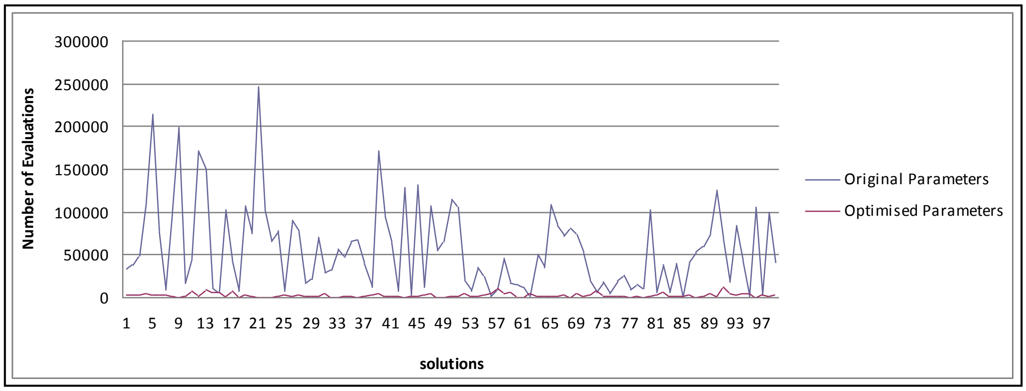
(a) De Jong function



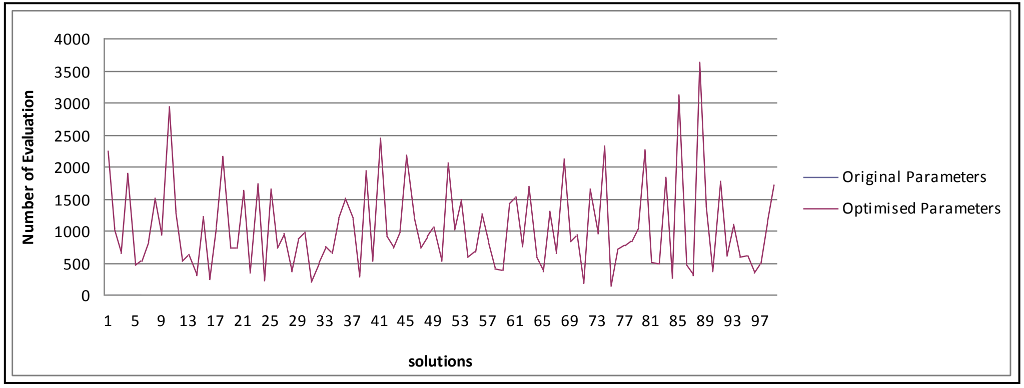
(b) Martin and Gaddy function



(c) Rosenbrock function 2D (a)



(d) Rosenbrock function 2D (b)



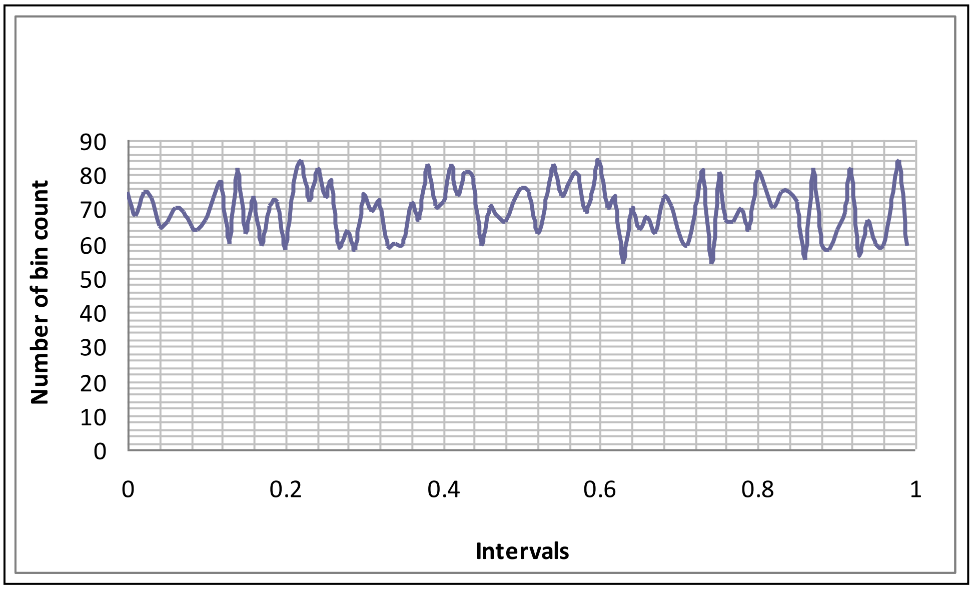
(e) Griewank function

Figure 3.12 (a-e) Student T-Tests for both the Bees Algorithm and Meta Bees Algorithm

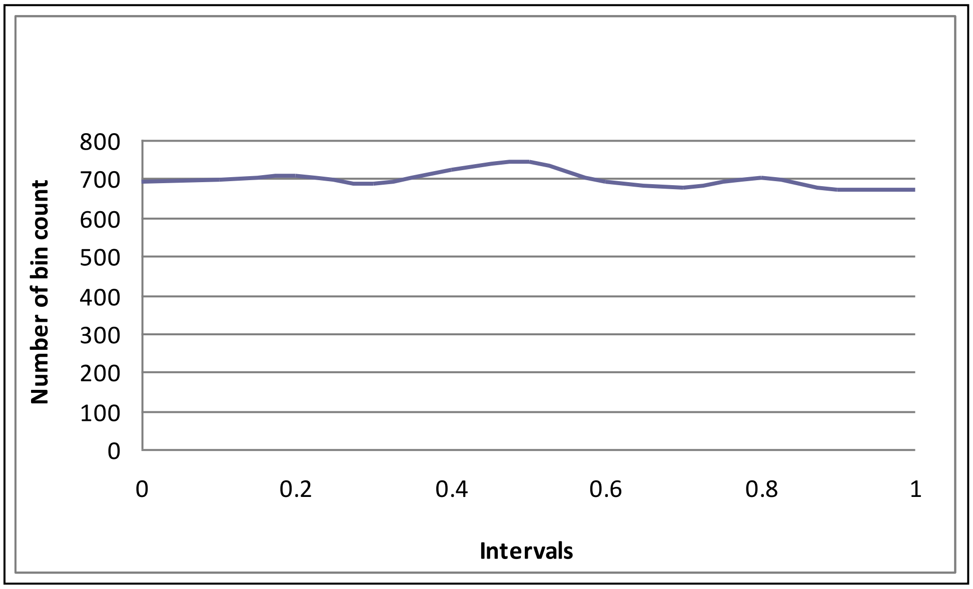
3.3.5.2 Random Distribution Test

A Random Distribution test was also performed to the both results obtained using the standard Bees Algorithm and Meta Bees Algorithm to ensure the quality of the random numbers generated by the system. The Bees Algorithm includes some randomisation in its two local and global searches such as the initial distribution of the scout bees over the search space and the recruitment of bees within the neighbourhood search area. This requires generating a sequence of random numbers. As the possible amount of generable random number is finite, after a certain number of calls the sequence of random numbers that has already been produced is repeated again. Hence, the larger this sequence length is, the better the number generator should be. Random number generators always need at least one integer value called the seed to get started. Different seeds do not usually lead to different sequences of random numbers, but the random number generator starts at different points in its finite sequence of random numbers which often leads to different results. Various tests have been developed to check the quality of randomness of these random number generators. The Bees Algorithm uses a uniform distribution of the random number which means that every point of the search space has the same probability of being chosen and not being biased to certain parts of the search space otherwise the randomness of the algorithm would be meaningless which will affect the mechanism of how the Bees Algorithm explores new solutions as discussed in chapter 3.

To test the quality of the random numbers a normalised distribution test was used (Schneider and Kirkpatrick, 2006). To visualise the distribution in this test, the interval is divided into certain number of subintervals called bins and then how many numbers there are in each bin are displayed. Figures 3.13 to 3.17 show two series of histograms that show the distribution of the generated random numbers during the Bees Algorithm search. One histogram is divided into 10 bins and the other is divided into 100 bins. For the value of bins equal to 10, some intervals get more random numbers than others, but as bins grow, the distribution of the random numbers becomes more and more equal among the intervals and graphs are smoother thus more acceptable randomness in the generated random numbers is provided for each benchmark.

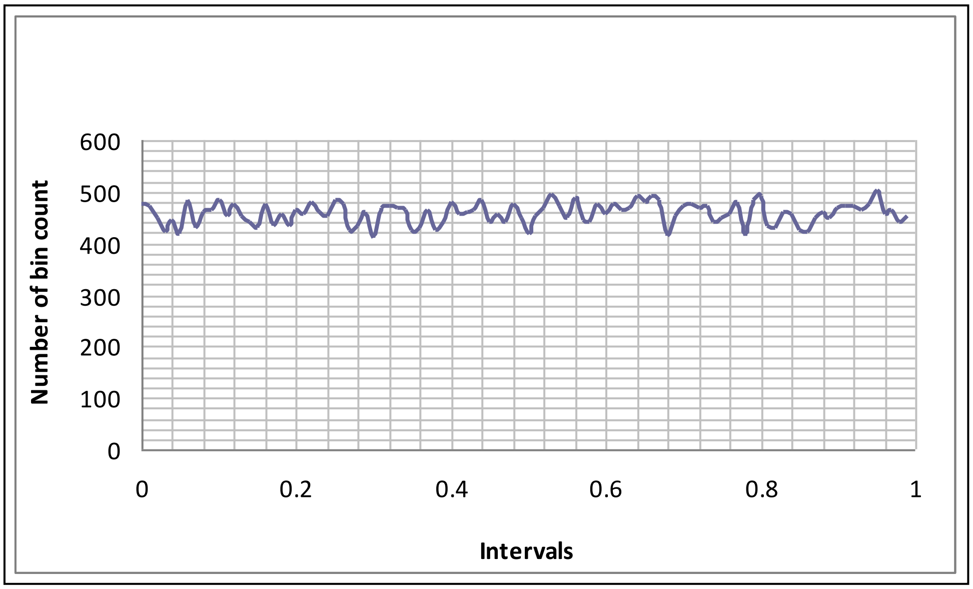


(a) 10 bins

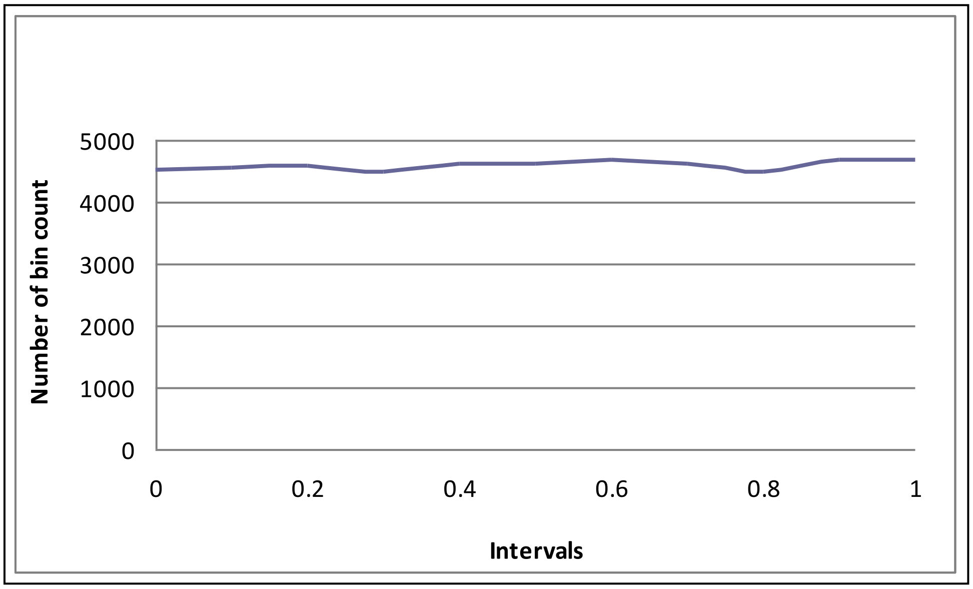


(b) 100 bins

Figure 3.13 Distribution of Random Numbers in both 10 and 100 Bins for the De Jong Function

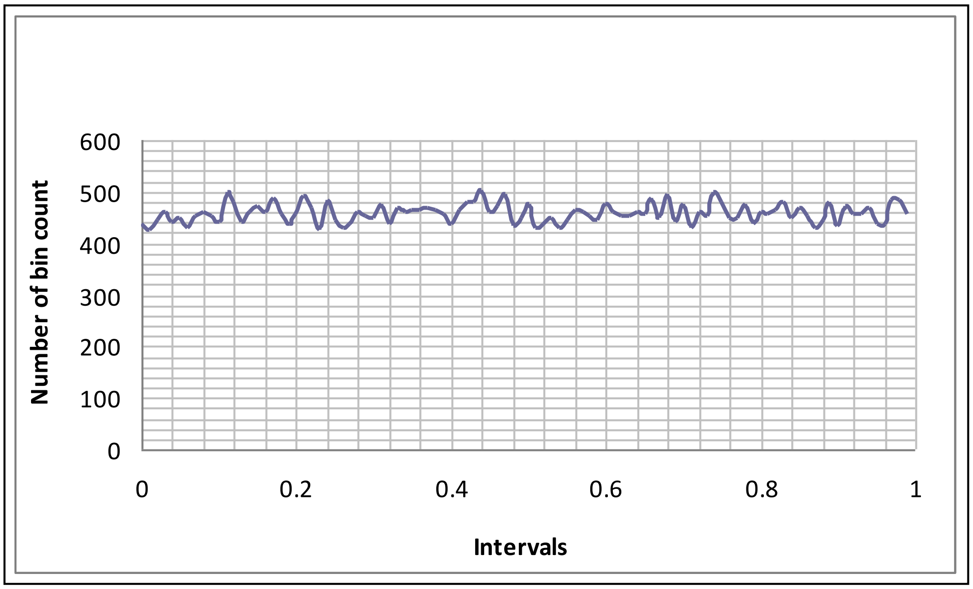


(a) 10 bins

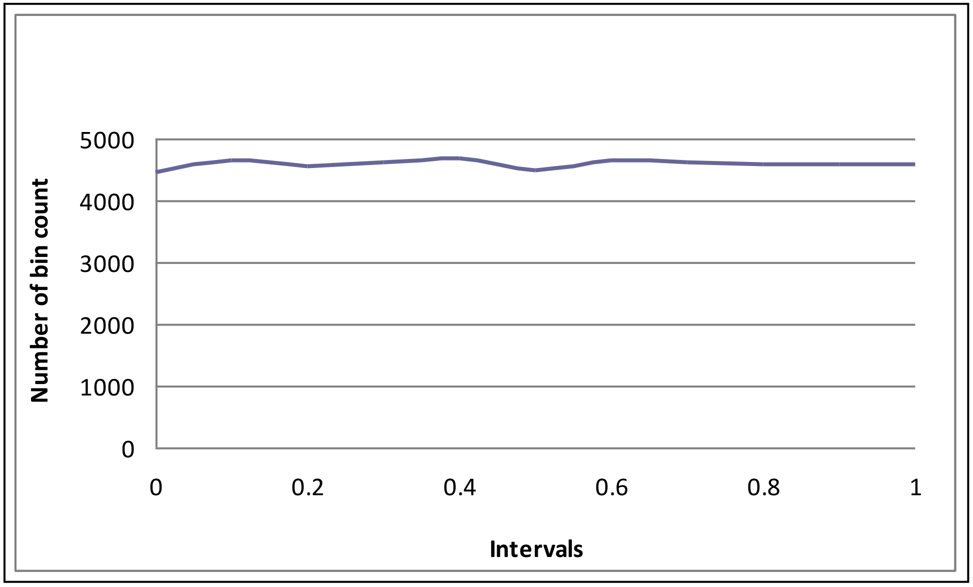


(b) 100 bins

Figure 3.14 Distribution of Random Numbers in both 10 and 100 Bins for the Martin and Gaddy Function

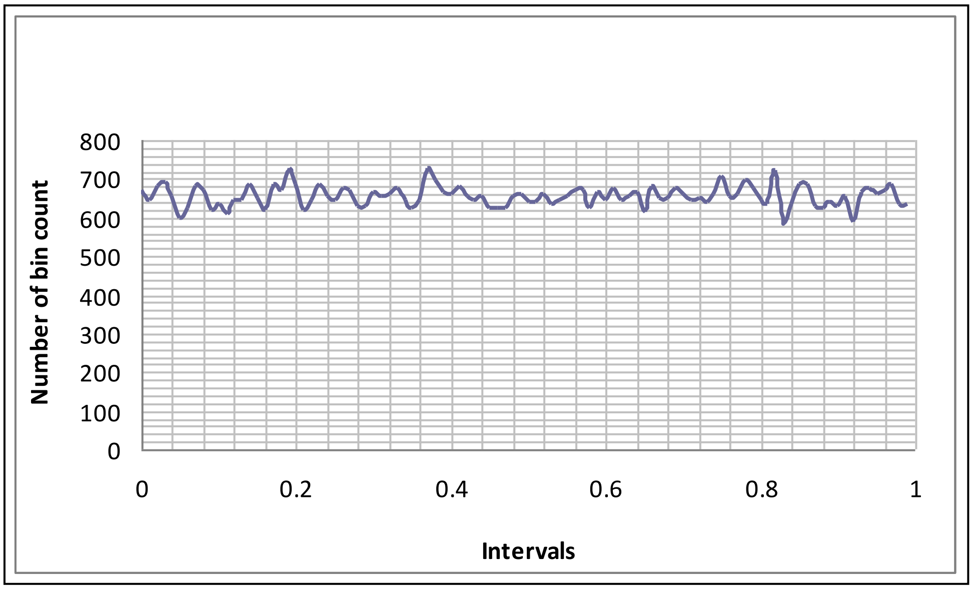


(a) 10 bins

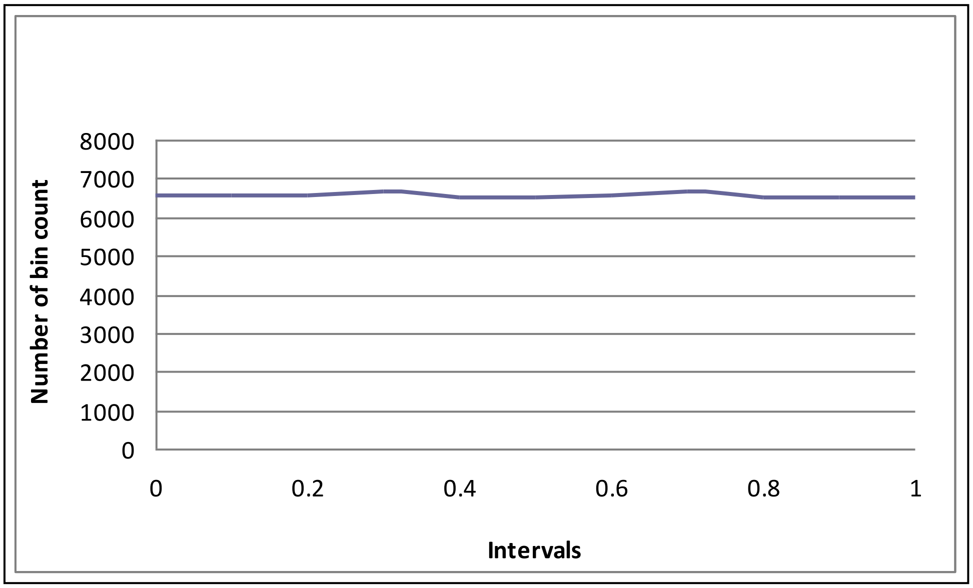


(b) 100 bins

Figure 3.15 Distribution of Random Numbers in both 10 and 100 Bins for the Rosenbrock (a) Function

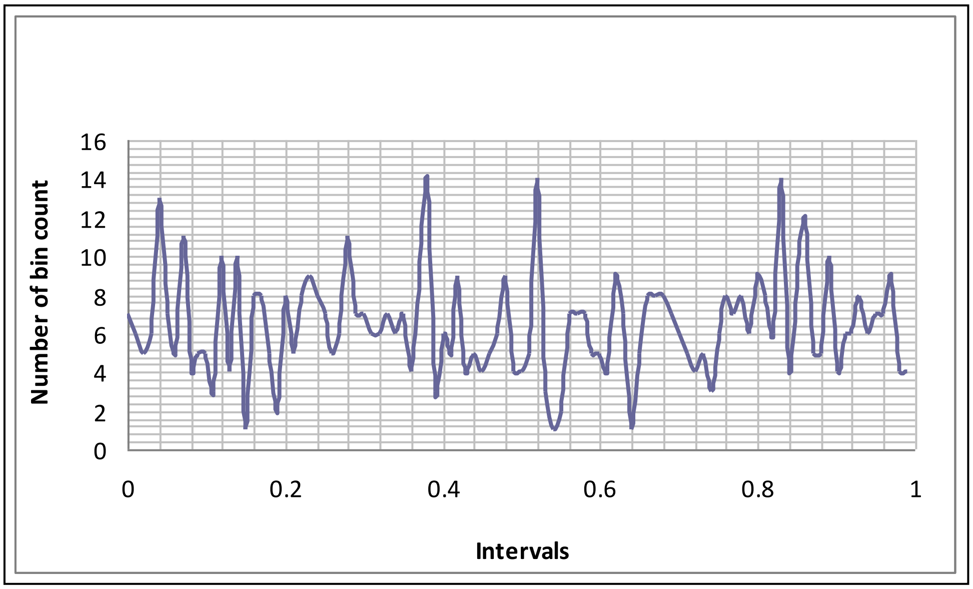


(a) 10 bins

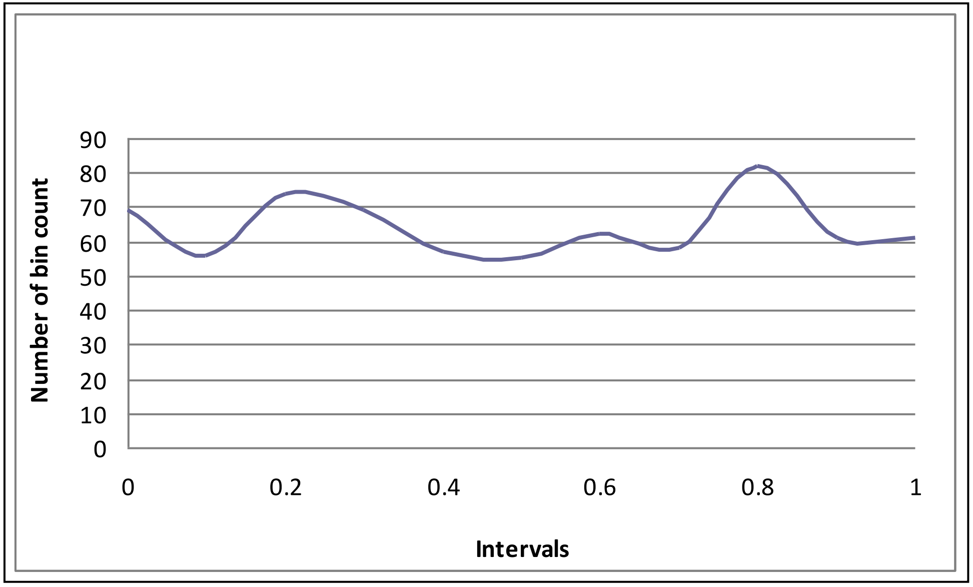


(b) 100 bins

Figure 3.16 Distribution of Random Numbers in both 10 and 100 Bins for the Rosenbrock (b) Function



(a) 10 bins



(a) 100 bins

Figure 3.17 Distribution of Random Numbers in both 10 and 100 Bins for the Griewangk Function

3.4 SUMMARY

The aim of the study reported in this chapter was to characterise the behaviour of the Meta Bees Algorithm, highlight its strengths and weaknesses and show the differences between the performance and reliability of the proposed method and those of competing algorithms.

The search strategy of the Bees Algorithm is to combine global random exploration with local neighbourhood sampling. Explorative search (using scout bees) and exploitative search (employing recruited foragers) are clearly differentiated and they can be independently varied through a set of learning parameters. This clear decoupling between exploration and exploitation facilitates the tuning of the Bees Algorithm.

Parameter tuning for metaheuristic search algorithms can be a time-consuming and inexact way to find appropriate parameter values to suit various classes of problems. An alternative approach has been explored in this chapter in which the algorithmic mechanics of the Bees Algorithm are used to produce suitable values while problems are being solved.

The performance of the Meta Bees Algorithm was evaluated using five benchmark minimisation tasks. The results were compared with those produced by the Standard Bees Algorithm, the SIMPSA, the NE SIMPSA, the GA and ACS.

The results for the function optimisation instances used to test this notion suggest that its performance, in terms of solution costs and run times, is comparable to the standard implementation in which values from (Pham et al., 2006b) were employed. In fact the performance in terms of objective cost was often an improvement over the standard set. This may be attributed to the ability of the new Meta Bees Algorithm to tailor parameter values to the problem instance being solved.

The results reveal that the improved solver generally performs well against one that uses standard/fixed parameter values. This is attributed to the fact that parameter values suitable for a particular problem instance can be automatically derived and varied throughout the search process.

The results also highlight the importance for the Bees Algorithm to conduct a sustained and thorough exploitation of the parameter search space. In addition, the results show that good quality solutions are achieved for a range of function optimisation problems.

The tests proved the strength of the Bees Algorithm in terms of accuracy, learning speed and robustness. In four of the five benchmark cases considered, the Bees Algorithm ranked amongst the top performing optimisation procedures.

The results reveal that for the function optimisation problem, the use of the Meta Bees Algorithm solver generally performs well against one that employs a standard set of parameter values. This is attributed to the fact that parameter values suitable to a particular problem instance can be automatically derived and varied throughout the search process.

CHAPTER 4

MODIFIED BEES ALGORITHM FOR SOLVING STOCHASTIC OPTIMISATION PROBLEMS

This chapter describes the application of the Bees Algorithm and statistical analysis to the solution of a stochastic optimisation. The algorithm employed to carry out this task was designed with a new fitness evaluation method based on computing the average fitness value for each bee over a number of trials rather than computing the value of a single trial. The method enabled the algorithm to be applied to those situations where the returned value of the fitness function of a site is different every time this function is called. Hence, there is a need to evaluate this site a number of times by sending a number of bees to the same site (point) and then calculating their average value. This means that the number of bees sent to each site is not constant and it is based on feedback information gathered from the bees recruited so-far to the same point. To test the algorithm, the parameter value optimisation of a metaheuristic method is employed. This test took the shape of parameter setting for the ACS algorithm which is employed to solve a TSP as a stochastic problem.

The chapter is organised as follows: the new modified Bees Algorithm for solving stochastic optimisation problems is presented in the next section. Section three details the analysis of the algorithm and the steps that it takes to converge to the near optimum solution. The application of the algorithm to parameter setting for the ACS-TSP problem and a description of the experimental conditions concludes the chapter.

4.1 PROPOSED MODIFIED BEES ALGORITHM FOR SOLVING STOCHASTIC PROBLEMS

This section proposes a modified Bees Algorithm for solving stochastic problems. The algorithm works like the original Bees Algorithm (see Figure 3.1) but with the inclusion of a new statistical part which leads the Bees Algorithm to evaluate the performance of each bee (see Figure 4.1). The original Bees Algorithm involves calling the fitness function of the site where the bee has been placed either randomly or directly within the patch. In deterministic problems such as those the Bees Algorithm has dealt with before, the value of the fitness function is fixed for the same site visited by a bee and hence evaluating it only once is enough. However, with the new algorithm, the modified Bees Algorithm is tested on stochastic problems where the fitness function consists of a set of parameters which change randomly. This means that every time the fitness function is called, its return value will be different so there is a need to evaluate each site a number of times by sending a number of bees to the same site and then calculating their average value. The number of bees sent to each site is not constant and it is based on feedback information gathered from the bees recruited so-far to the same point. The feedback information that will determine the number of bees sent to the same point is based on the slope of the line formed from the linear regression of the fitness values of the sent bees. The algorithm will stop sending more bees to the same site when it indicates a near horizontal line. By adopting this technique, a more accurate average fitness value for the sent bees to the same site is guaranteed while at the same time consuming the minimum number of evaluations. The accuracy of this average value will depend on the minimum number of bees sent to the same point and the minimum acceptable slope value.

Figure 4.1 shows the pseudocode of the new algorithm. It is exactly like the standard Bees Algorithm, the only difference being in the way that a bee is evaluated. A subroutine replaces the direct evaluation method. The acceptable slope value will determine the accuracy and the speed of convergence of the algorithm. If the acceptable slope value is very small, the algorithm will take a longer time to converge but its accuracy will be very high. On the other hand, if a higher acceptable slope value is chosen, the running time will drop, but so will the accuracy.

```
    1. Initialise population with random solutions.
    2. Evaluate fitness of the population by calling the subroutine.
    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
       fitness by calling the subroutine.
    6. Select the fittest bee from each patch.
    7. Assign remaining bees to search randomly and evaluate their fitness.
    8. End While.

// Subroutine for bees fitness evaluation.
    1. Evaluate fitness.
    2. While (minimum slope value not reached)
    3. Evaluate the fitness and store its value.
    4. End while.
    5. Calculate the mean from the stored values and return it.
    6. End Subroutine.
```

Figure 4.1 Pseudocode of the Modified Bees Algorithm for Solving Stochastic Problems

4.2 COMPUTATION EXPERIMENTS

The experimental analysis was conducted in three stages; the first stage was to answer the question of why there was a need to evaluate the same point in a search space more than once in a stochastic problem. To address this, a study on the parameters of the ACS and their influence on the speed of convergence of the TSP instance was performed. Secondly, a study and analysis was conducted on when to stop the evaluation of a point. This was achieved by studying the relationship between the slope of the line formed from the linear regression of the average fitness values of the sent bees and the accuracy of these values obtained by the algorithm. Finally, based on these two previous stages, the proposed method was applied on the ACS to find the best combination of parameters that guarantees the fastest convergence to the best solution on a TSP instant. The stages outlined above are now discussed in detail.

4.2.1 Analysis of the Speed of Convergence of ACS

The speed of convergence is represented graphically in the following figures for two different experiments with different sets of parameter values. Figure 4.2 shows the first experiment on the variation of the number of function evolutions that the ACS uses to find the optimum solution of a TSP instance called

Oliver30 (Oliver et al., 1987) and is illustrated in Figure 4.3 over 50 runs using the same set of parameters shown in Table 4.1. Statistical information, see Table 4.2, shows that the average value of the fitness which represents the value of speed of convergence is 463.

Figure 4.4 shows the second experiment using a different set of parameters as shown in Table 4.3 for the same TSP instance. Table 4.4 shows statistical information from this experiment where the average value over 50 runs is 2370.

Table 4.1 Parameters employed in the First Experiment

m	β	ρ	q_0
11	4	0.2	0.7

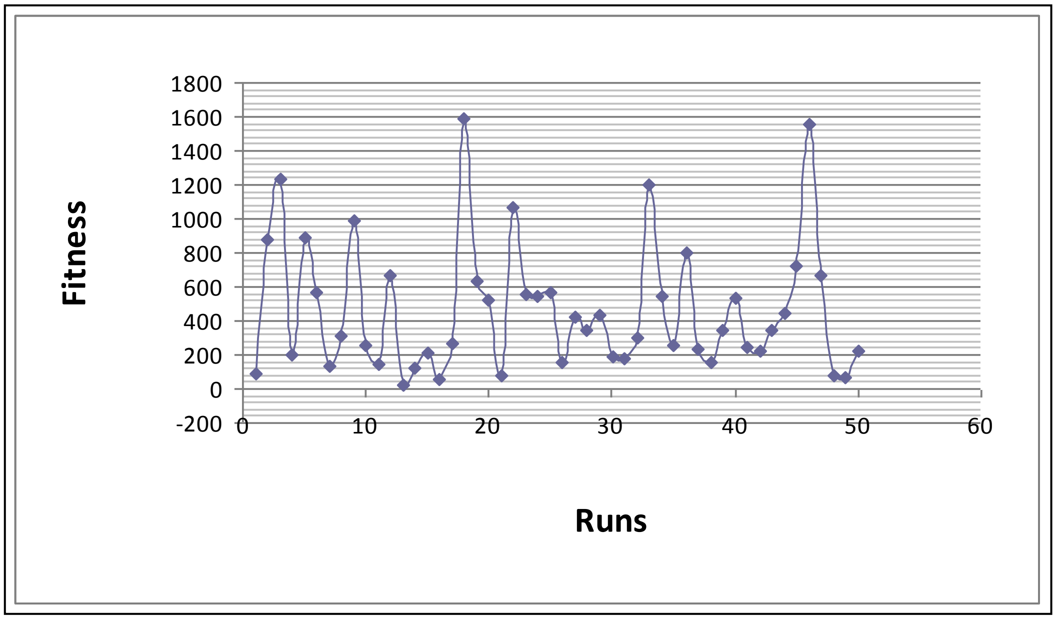


Figure 4.2 Fitness Values for the First Experiment over 50 Runs

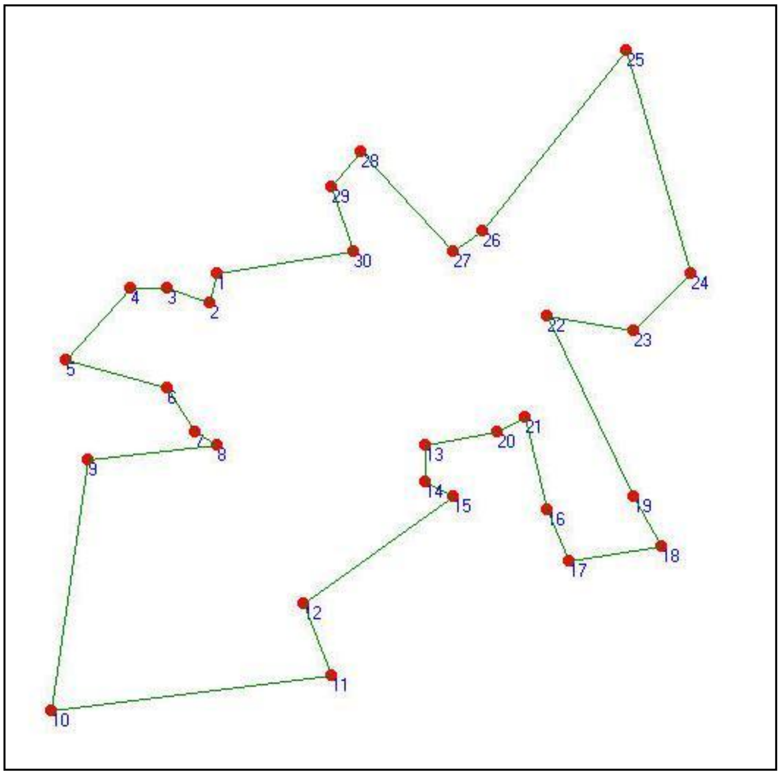


Figure 4.3 Oliver30 TSP Problem

Table 4.2 Statistical information for the first experiment

Runs	Min	Max	Average
50	22	1584	463

Table 4.3 Parameters employed in the Second Experiment

m	β	ρ	q_0
25	13	0.7	0.2

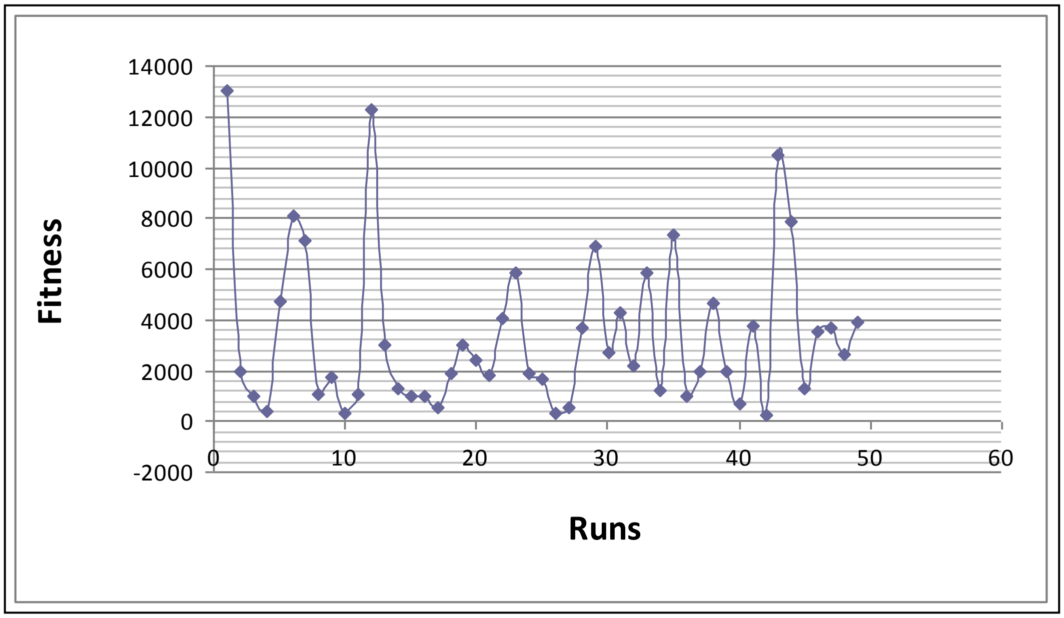


Figure 4.4 Fitness Values for the Second Experiment over 50 Runs.

Table 4.4 Statistical Results from the Second Experiment

Runs	MIN	MAX	Average
50	275	13050	2370

The above two experiments demonstrated:

1. The importance of tuning the parameters of the algorithm to generate a higher quality solution (fitness function) because for every set of parameters totally different statistical values (minimum, maximum and average) are produced, and

2. Although the same set of parameters were employed for each of the two experiments over 50 runs each, the fitness value was never the same. This indicates, unlike with deterministic problems, the importance of evaluating the same point (a set of parameters in this case) several times and hence the need for the proposed method. In summary, this revealed that the parameter setting for the metaheuristic method can be considered a stochastic optimisation problem and it is affected directly by its parameters values.

4.2.2 Analysis of Calculating the Average Fitness

To demonstrate the relationship between the slope of the line formed from the linear regression of the average fitness values of the sent bees and the values obtained by the algorithm, an analysis was performed on a stochastic optimisation problem. Figures 4.5, 4.6 and 4.7 show three different variations of fitness values obtained from many evaluations of the same stochastic problem. Figure 4.5 shows the variation of fitness values for five runs. In this figure the slope is not equal to zero, which means more evaluations and fitness values are needed to obtain a more accurate average value. This means that five evaluations are not sufficient to calculate the precise average value.

Figure 4.6 shows the variation of fitness values obtained from evaluating the same stochastic problem over 22 runs, the slope of the line formed is near to zero and the line is nearly horizontal. When the average value of slope of the line for 22 runs is compared to the average of 90 runs, Figure 4.7, it shows that they are nearly the same.

The slope of the line formed by applying a least-squares liner regression on the bees sent to the same point is an indication of the number of evaluations that is required of this particular point to be able to represent a more accurate average value of its fitness. By using this type of feedback in the modified Bees Algorithm, it will ensure having a more accurate average value while requiring only a minimum number of evaluations. In conclusion, adding the statistical feature to the Bees Algorithm allows it to solve stochastic optimisation problems.

Table 4.5 shows the average obtained for three different numbers of runs; 5, 22 and 90 runs, where the average values found were 658, 472 and 471 respectively.

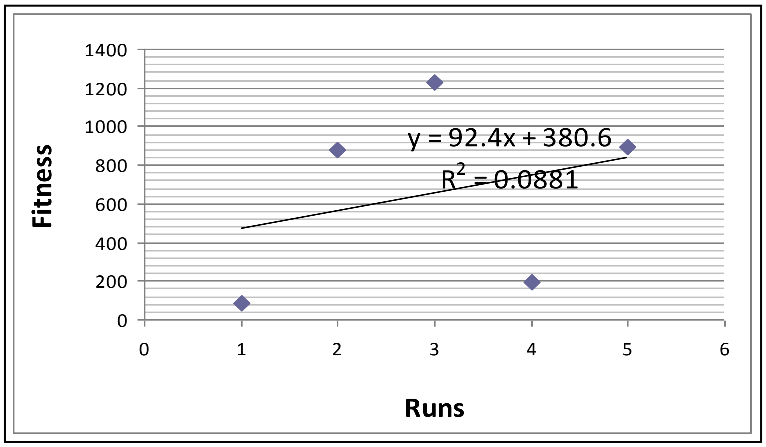


Figure 4.5 Non-Zero Slope of Regression Line Shows an Insufficient Number of Evaluations for Calculating the Average Value

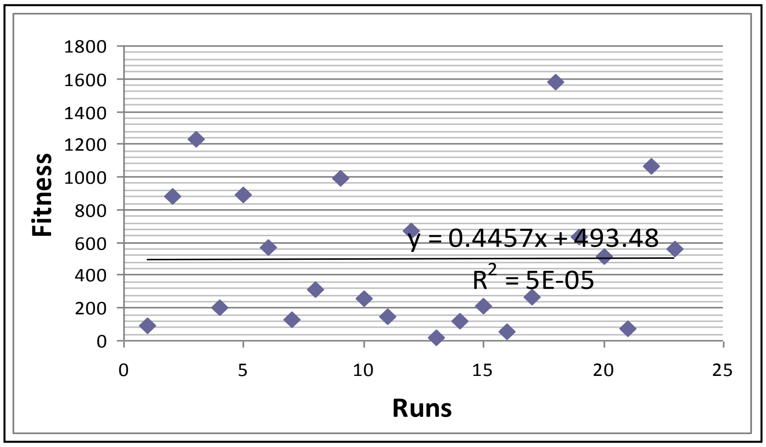


Figure 4.6 Number of Evaluations - a Representation for 22 Runs

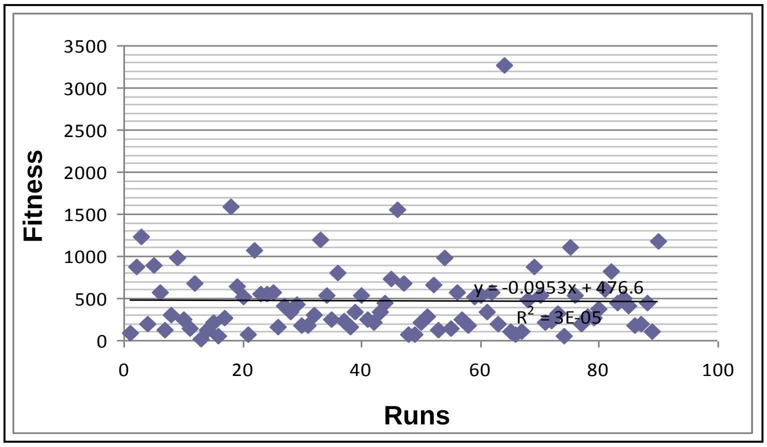


Figure 4.7 Number of Evaluations - a Representation for 90 Runs

Table 4.5 Fitness Values for 5, 22 and 90 Runs

Number of Runs	Fitness Value		
	MAX	MIN	Average
5	1232	88	658
22	1584	22	472
90	3267	22	471

4.2.3 The Proposed Bees Algorithm

The problem of setting parameters can now be solved using the modified Bees Algorithm. To examine the efficiency of the modified Bees Algorithm, it is tested on the problem of finding the best parameter values for the ACS by solving TSP instances.

The fitness value is the total number of evaluations consumed by the ACS to find the optimum solution of the TSP instance (see Figure 4.8).

The ACS parameters chosen to be optimised are the number of ants (m), the parameter that determines the relative influence of the heuristic value (β), the parameter that balances between exploration and exploitation behaviours of the algorithm (q_0) and the parameter that governs the local pheromone evaporation (φ). The search spaces for these parameters are shown in Table 4.6. The other parameters (the length of the candidate list (cl = 10), the parameter that governs global pheromone decay (φ=0.1) and the initial value of the pheromone trails value (τ_0) are initialised by their default values as mentioned in section 2.4.6.3.

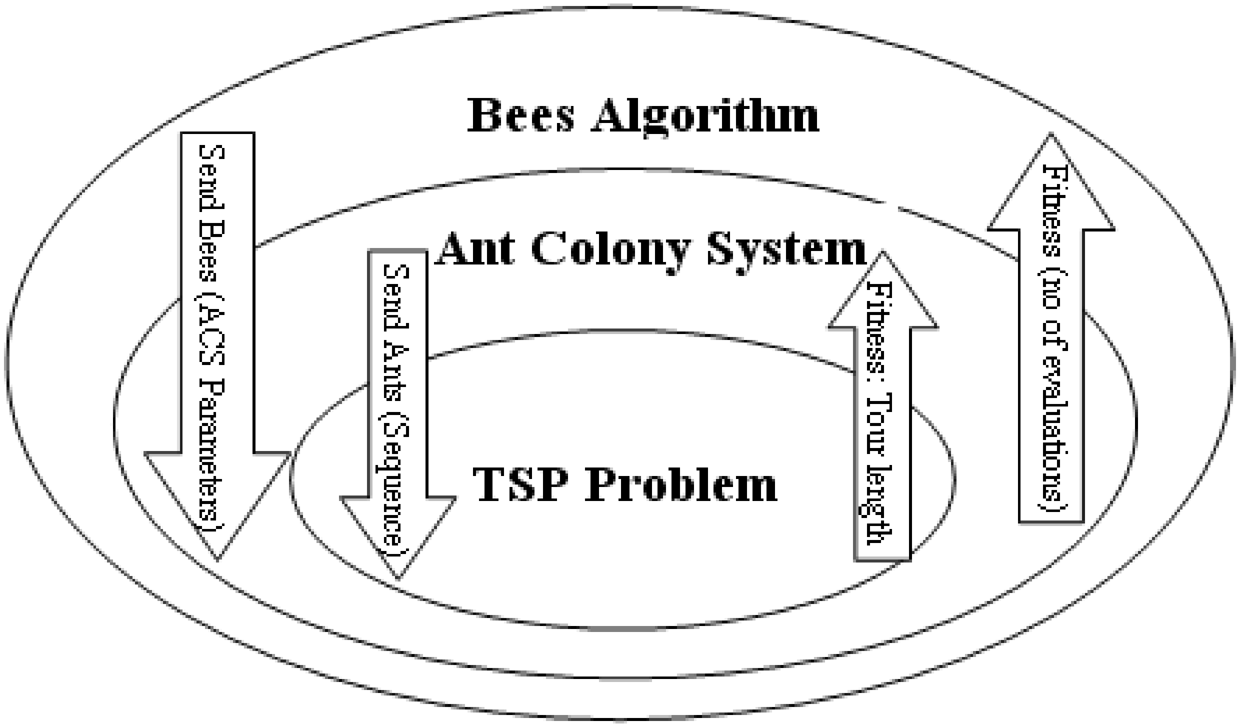


Figure 4.8 Layers Showing the Bees Algorithm Wrapping the ACS-TSP

Table 4.7 shows the best parameters obtained by applying the modified Bees Algorithm on the ACS-TSP Oliver30 dataset. A comparison of the results with other research found in the literature is also presented.

Experiments were produced on a 2.00GHz Pentium Dual Core processor with 1GB of RAM. The process took three days to arrive at the best parameters for the Oliver30 dataset using the programming language C#.Net 2008 and the .Net framework 3.5.

Table 4.6 Search Spaces of the ACS Parameters

Parameter	m	β	ρ	q_0
Min value	2	0	0	0
Max value	Number of Cities	14	1	1

Table 4.7 The Optimum Results Obtained from the Modified Bees Algorithm Compared to Suggested Values in the Literature

References	m	β	ρ	q_0	Mean no. of evaluations
(Dorigo and Gambardella, 1997)	10	2	0.1	0.9	7839
Pilat and White (2002)	20	6	0.2	0.7	7443
Modified Bees Algorithm	12	4	0.2	0.8	2984

4.3 SUMMARY

The chapter has described a method to solve stochastic optimisation problems. The key operation in this method was the multiple evaluation of the fitness of each parameter setting tried, when using a modified version of the Bees Algorithm to optimise stochastic problems based on statistical analysis. The new algorithm was tested on the problem of finding the best values of the parameters for an ACS-TSP.

The results obtained show that the algorithm was able to effectively find near optimum solutions of such stochastic optimisation problems. Also, the results show that parameter setting produced by the Bees Algorithm outperformed parameter sets that were suggested by other parameter settings methods such as the ACS and the GA

Further work could investigate the possibility of using the variation in the values of the fitness with a particular parameter setting as an indicator of the stability of that setting. For such an investigation, it might be useful to adopt a multi-objective optimisation technique such as that presented in (Ghanbarzadeh, 2007). Also, this modified Bees Algorithm can be applied to many well-kown TSP benchmarking instances for finding the best parameters for them by taking the average of these optimum parameters for different instances.

CHAPTER 5

APPLICATION OF THE BEES ALGORITHM TO THE SOLUTION OF COMBINATORIAL OPTIMISATION PROBLEMS

This chapter discusses the use of the Bees Algorithm to solve combinatorial optimisation problems. The problems belong to the general class of Travelling Salesman Problems. In particular, the application studied is that of optimising the scheduling of operations for a printed circuit board (PCB) assembly machine.

5.1 BEES ALGORITHM CHARACTERISTICS FOR COMBINATORIAL OPTIMISATION PROBLEMS

In general, the Bees Algorithm is a population-based stochastic search algorithm designed to solve specific types of optimisation problems (Ghanbarzadeh, 2007). When the Bees Algorithm is applied to combinatorial optimisation problems, these problems are generally characterised by the following:

- the search space is discrete;

- there is a set of finite constraints;

- a solution is represented as an ordered sequence of components;

- there is a cost function which associates a cost to each solution generated by the search algorithm, where each component added to a solution contributes to the total cost of the solution;

- there is a finite set of components from which solutions are constructed;

- there is a finite set of possible transitions among the complements;

- there is a finite set of sequences of components representing all possible valid combinations of components.

Given these characteristics, the purpose of the Bees Algorithm is to construct a feasible sequence of components such that the cost of the solution is minimised when implemented in the minimisation problem (Pham et al., 2007a) and such that the sequence of components represents only valid transitions.

The Bees Algorithm applied to combinatorial problems requires a graphical representation of these problems i.e. a graph that consists of a finite number of nodes and links between nodes (Engelbrecht, 2005). Each node represents one of the components, and a link represents a transition from one node to the next. A cost is associated with each link. The objective of the Bees Algorithm is to traverse this graph, in order to construct a minimum cost path. The constructed path represents a sequence of components, i.e. a solution to the optimisation problem. In the case of the Bees Algorithm, a path is constructed by randomly generating all the nodes at once.

The Bees Algorithm models a colony of honeybees searching for multiple solutions in parallel. To achieve the task of constructing optimal paths, bees have the following properties and characteristics:

- A bee has a memory to store information on the path constructed. The memory is employed mainly to enforce constraints, such as the inclusion of a component only once.

- One or more termination conditions are associated with each bee. These conditions include: ‘A solution with acceptable cost has been constructed’, ‘A maximum number of iterations has been exceeded’ or ‘Stagnation behaviour is observed’.

To demonstrate the characteristics of the proposed algorithm for solving combinatorial problems, the Bees Algorithm has been applied to the TSP as it is easy to understand, has numerous applications and has been studied extensively by researchers from various disciplines (Aarts et al., 1988; Chandra et al., 1999; Freisleben and Merz, 1996; Gambardella and Dorigo, 1996; Helsgaun, 2000;

Knox, 1994; Laporte, 1992). In addition, the TSP is considered difficult to solve as it is a non-deterministic polynomial-time (NP-hard) problem, which means that the time complexity of finding an optimal solution to these types of problems grows exponentially with the problem size (e.g. the number of cities) (Laporte 1992). In a TSP with N cities, there are $\frac{(N-1)!}{2}$ possible tours that must be computed in order to determine the optimal path. Rather than searching all possible tours to find an optimal solution, a common approach for this class of problem is to find a solution that is “good enough”.

Many of these “satisfying” solutions can be determined in polynomial time. The TSP is a problem of finding the shortest tour that visits all the nodes of a fully connected graph, the nodes of which represent locations, and the arcs representing paths with associated costs (normally assumed to be distance).

5.2 THE BEES ALGORITHM FOR SOLVING THE TRAVELLING SALESMAN PROBLEM

The TSP has been one of the most popular combinatorial optimisation problems studied. A TSP can be either symmetric or asymmetric. A TSP is considered symmetric if travelling from city A to city B is given the same weight as travelling from city B to city A. This would be the case if we only consider distance, however if other factors are taken into account then we may not be able to assign the same weight to travelling in both directions between two cities. Such cases are called asymmetric TSPs or ATSPs (Freisleben and Merz, 1996). There are numerous techniques employed for tackling this class of combinatorial optimisation problem, including methods that are specifically tailored to the TSP as well as more general-purpose metaheuristics which can be applied to a number of hard combinatorial problems such as the graph partitioning problem and the quadratic assignment problem (Garey and Johnson, 1979).

In the subsequent section, the main steps required to implement the Bees Algorithm when applied to solve the TSP will be presented.

5.2.1 Implementing the Bees Algorithm for the Travelling Salesman Problem

The TSP can be represented as a sequence of *N* cities to be visited, where the actual order of the sequence determines a particular solution to the problem (Johnson and McGeoch, 1997). Thus in general the search space consists of all *N!* permutations.

The Bees Algorithm can be applied to the TSP in a straightforward way. Where each bee represents a candidate tour and where the first and the last element of the bee represent the city of origin. For example, the bee *b* (B, D, A, E, C, B) represents a tour from city B, via cities D, A, E and C, then back to city B.

When applying the Bees Algorithm to a TSP as shown in Figure 5.1, the algorithm starts with an initial population of *n* scout bees randomly distributed in the search space by generating a set of sequences representing the visiting sequence of each bee. Each bee becomes a symbolic string representing the sequence of cities. For *N* cities to be visited by a salesman, a string with a length of *N* is needed to encode each candidate solution where each bee will visit (C) number of cities generated randomly from one to (C), which formulates (C) links employed as the initial population for each bee.

To generate *b*(B, D, A, E, C, B), a random number generator is employed to generate letters between A and E and once a letter is generated (i.e. B in this example) it will not be allowed to appear again. The first letter generated will appear again at the end to give a closed tour. Then the generator generates another letter (D in the above example), and so on until all the five letters are generated to constructed a full and closed tour.

In step 2, the fitness computation process (i.e. the performance evaluation of the candidate solutions) is carried out for each patch visited by each bee. This is completed by calculating the distance between each two adjacent cities in the sequence and adding all the distances together to find the total length of each tour using the objective function shown in Equation 5.1:

$$f(\pi) = \sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)}) \qquad \text{Equation 5.1}$$

Where d(c_{i},c_j) is the distance between cities *i* and *j*, and π(*i*) for *i=[1], N* defines a permutation.

For the next step, step 3, the *m* patches (i.e. the tours) with the highest fitness levels are designated as “selected patches” and chosen for a neighbourhood search.

Once completed, step 4, is commenced. The algorithm searches around the selected patches using a local search algorithm. Various operators could be employed to create neighbours to a given bee, including monadic operators such as mutation, inversion, swap and insertion (single or multiple). For the test problems considered in the next section, the single-point insertion and the two-paths optimal (2-Opt) operators (Burke et al., 1999) are adopted and shown.

In step 5, each tour created by a local operator represents a neighbour to the selected patches, and more tours will be created for the best *e* patches. In step 6, only the bee with the highest fitness (lowest tour length) in each patch will be selected to form the next bee population. In step 7, the remaining bees in the population are placed randomly around the search space to scout for new potential solutions (tours).

Steps 4-7 are repeated until either the best fitness value (which will be designated as the ‘global best tour’) has been found or the specified maximum number of iterations has been reached.

At the end of each iteration, the colony will have 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.

```
1- Initialise population with random solutions.
2- Evaluate fitness of the population.
3- While (stopping criterion not met)
        //Form new population.
4- Select sites for a neighbourhood search using a neighbourhood operator.
5- Recruit bees for selected sites (more bees for best sites) and evaluate fitness.
6- Select the fittest bee from each patch.
7- Assign remaining bees to search randomly and evaluate their fitness.
8- End While loop.
```

Figure 5.1 Pseudocode of the Standard Bees Algorithm

5.2.2 The Bees Algorithm with Neighbourhood operators

Given a feasible solution, x, to the problem, its neighbourhood is a set of solutions obtained from x by disturbing its components in some specified manner (Ibaraki, 1997). Two neighbourhood operators for the TSP have been implemented for the Bees Algorithm. These are:

A. Single-point insertion

Illustrating this process with an example, a closed sequence C B E A D C shown in Figure 5.2a, will be modified in the following way. The insertion operator breaks up the sequence by removing a section, in this case BE, between two randomly chosen positions B and E. This section is then inserted at a randomly selected point within the sequence; in this case, it is placed after point D. It must be noted that this is carried out while preserving the visiting order of the positions on the inserted section, producing the new sequence: C A D B E C. The sequence is then reconnected, as shown in Figure 5.2b.

Original closed sequence :	C B E A D (C)
Sequence after removal of section (BE) :	C A D (C)
Sequence following insertion of (BE) after point D :	C A D B E (C)

Figure 5.2a Single-Point Insertion

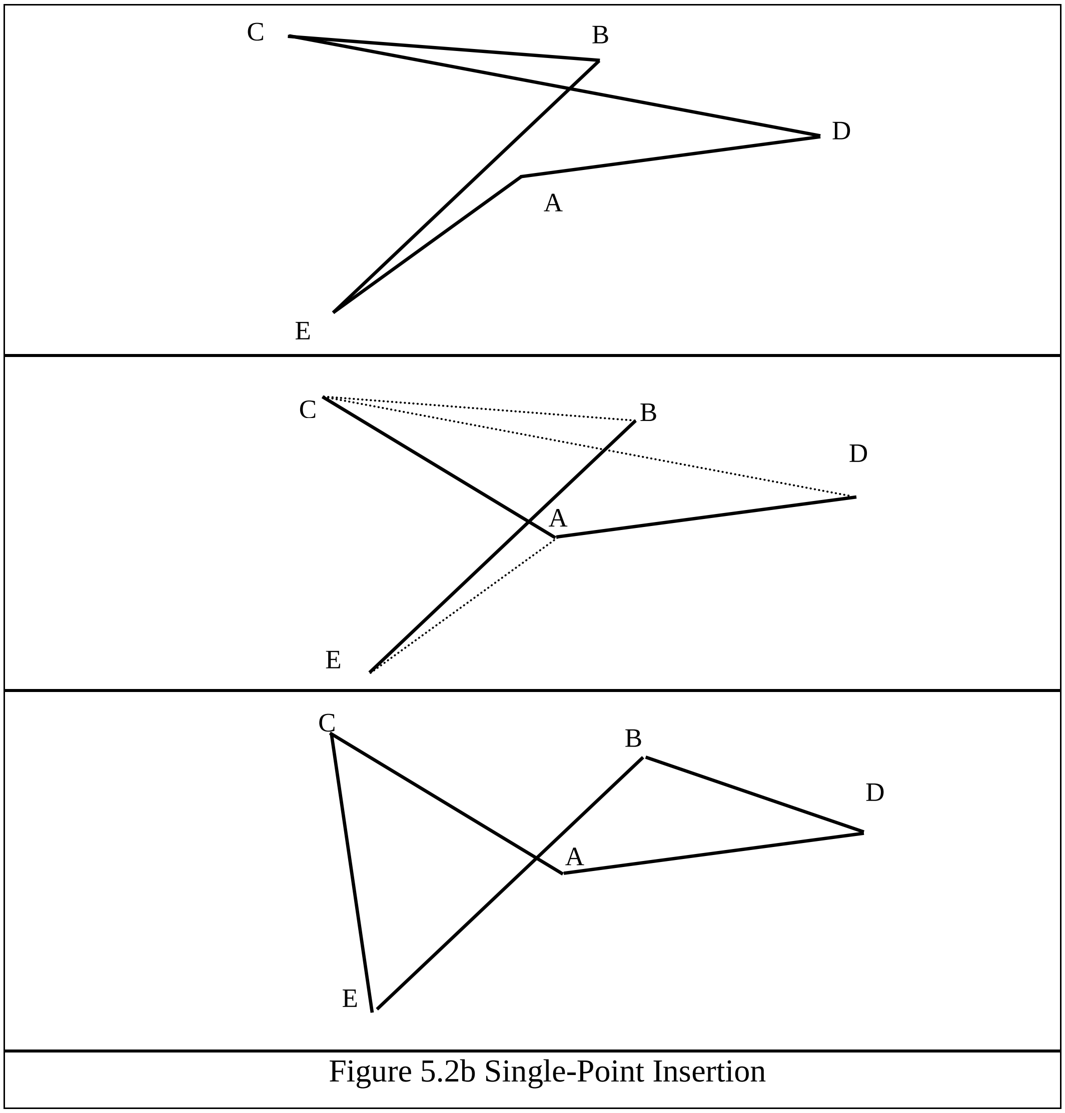


Figure 5.2b Single-Point Insertion

B. The 2-Opt operator

The two (two paths) optimal (2-Opt) algorithm is a tour improvement procedure (Okano et al., 1999). The best-known tour improvement procedures have been found to be edge exchange procedures (Lin and Kernighan, 1973) such as the k-Opt algorithm where all exchanges of k edges are tested until there is no feasible exchange that improves the current solution; this solution is said to be k-optimal. Since the number of operations increases rapidly with increases in k, k = 2 and k = 3 are most commonly employed.

A 2-Opt operator involves randomly breaking a sequence into two paths. The visiting order of the positions (cities) on one of the paths is then reversed before the two paths are reconnected. The application of a 2-Opt operator to a closed sequence of five positions (cities) can be seen in Figure 5.3a below.

In addition to the use of single-point insertion, the 2-Opt local search algorithm will be applied to the Bees Algorithm for the TSP. In the 2-Opt algorithm, two tours are considered neighbours if one can be obtained from the other by deleting two edges, reversing one of the resulting two paths, and reconnecting them. In the TSP, the 2-Opt neighbours of the tour are pair-wise part exchanges within the current solution sequence. For example, with a number of cities n = 5 and an original solution sequence of (C, B, D, A, E) shown in Figure 5.3b, it will be modified in the following way. Two edges will be deleted randomly (e.g. BD and EC), thus breaking the sequence into two paths (CB) and (DAE). One of the paths (DAE) will be reversed to become (EAD) and then the two paths will be reconnected. The new sequence is C B E A D. The sequence is then reconnected.

Original closed sequence:	C B D A E (C)
Broken-up sequence:	C B -- D A E
Path 1:	C B
Path 2:	D A E
Path 2 (after reordering):	E A D
Re-connected sequence:	C B E A D (C)

Figure 5.3a 2-Opt Operator

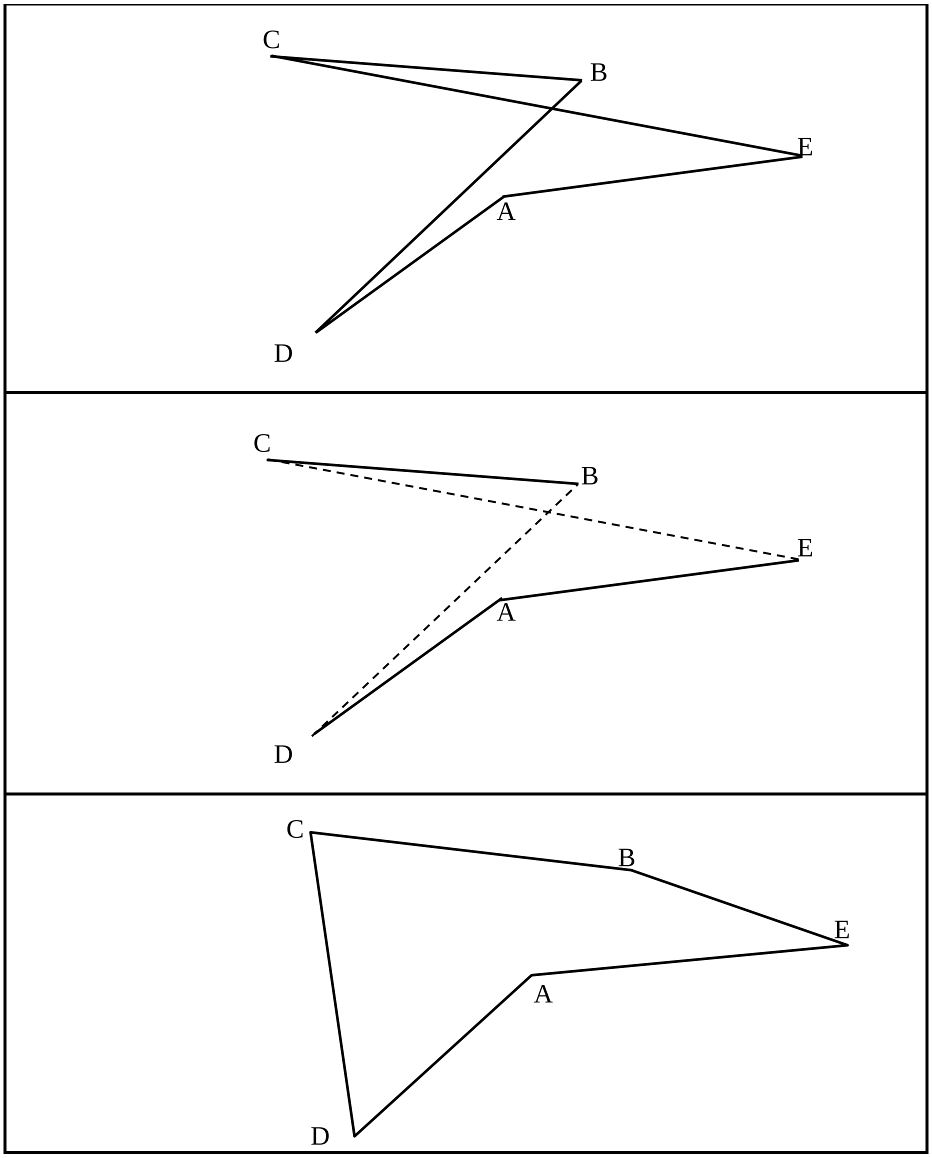


Figure 5.3b 2-Opt Operator

In addition to the random way, there are different ways to generate a starting tour for the initial application of the 2-Opt algorithm. A typically way is to use a random number generator. In addition, a greedy solution obtained by a greedy algorithm can be employed. For example, applying a greedy algorithm (such as the Nearest Neighbourhood algorithm) to the TSP yields the following instruction: "At each stage visit the unvisited city nearest to the current city" (Lawler et al., 1985). In this study however, the starting point is rapidly attained using the Bees Algorithm as explained in Section 5.2.1.

5.2.3 Experiment Results using the Bees Algorithm

For the purpose of comparison with other optimisation techniques, the Bees Algorithm was tested on three specific benchmark TSPs. The dimensions of these problems were 16, 44 and 91 cities with optimum solutions equal to 377.006mm, 569.706mm and 581.421mm respectively. The Euclidean metric which is the distance between two points is employed in these examples. Table 5.1 presents the results produced by the Bees Algorithm along with those produced by three other approaches for the same TSPs. These approaches are the GA, SA and Ant Colony System (ACS) (Stützle and Hoos, 1997).

Table 5.1 Performance of the Bees Algorithm Compared to the GA, SA and ACS

The algorithm	TSP instances					
	16 Cities		**44 Cities**		**91 Cities**	
	No. of iterations	Tour length	No. of iterations	Tour length	No. of iterations	Tour length
GA	269	377.006	1908	593.689	4780	721.074
SA	4810	377.006	97392	670.938	307921	884.009
ACS	213	377.006	**9**	569.706	7	581.421
Bees Algorithm	**190**	377.006	450	**569.706**	500	669.096

The results show that the Bees Algorithm is able to find the optimal solution for two of the benchmark problems (16 cities and 44 cities) and a near optimal solution to the third problem. The Bees Algorithm required less time (a fewer number of iterations) to find the solution to all three problems than either the GA or SA. However, while it could outperform the ACS for the smallest sized problem (16 cities), it needed more time for larger numbers of cities. Table 5.1 also shows that the performance of the Bees Algorithm generally drops when the size of the problem instances increases. One of the reasons could be the parameter settings that need to be fine-tuned to cater for different scenarios.

The next part of this chapter discusses the use of the Bees Algorithm to solve more complex optimisation problems than the TSP and that is the Printed Circuit Board (PCB) assembly planning problem. This is carried out over four sections:

The first part, Section 5.4.1, reviews the basic types of PCB assembly machines available and the assembly planning problems to which optimisation techniques have been applied. The second part, Section 5.4.2, describes the general use of the Bees Algorithm to generate optimal PCB assembly plans. The third part, Section 5.4.3, demonstrates the application of this algorithm to one type of placement machine. The final part, Section 5.4.4, presents the results obtained.

5.3 THE BEES ALGORITHM AS APPLIED TO THE PRINTED CIRCUIT BOARD ASSEMBLY OPTIMISATION PROBLEM

PCB assembly is the process of placing electronic components of different shapes and sizes employing various types of surface mount technology (SMT) placement machines at specific locations on a PCB. These placement machines have the ability of fast component placement and can handle high and rapid production demands. In its simplest form, PCB assembly optimisation aims to minimise the time needed to process the different components. Similar to the TSP, this time grows exponentially with the problem size (i.e. the number of components).

In order to make full use of the high speeds of these placement machines, many intelligent optimisation techniques have been applied to find the best-solution to the PCB assembly problem by optimising their operation. These optimisation techniques include GAs (Goldberg, 1989), SA (Mathur et al., 2000), and Evolutionary Programming (EP) (Nelson and Wille 1995). Results have shown that these techniques are able to provide near-optimal solutions in a short period of time.

5.3.1 Printed Circuit Board Assembly

A. A PCB Assembly Machine

A PCB assembly machine in its generic representation comprises of three parts:

- a feeder F_1 (or an array of feeders F_2) which supplies components,
- an assembly head H which picks up components from F_1 or F_2 and places them onto the PCB,
- a table T which carries the PCB.

Table 5.2 illustrates three types of machines for placing through-hole and surface-mount components in sequence onto a PCB (Ayob et at., 2002).

In a type–1 machine, a single feeder F_1, which can take the form of a magazine, provides components to a single assembly head H (there is no turret) at a fixed location in the horizontal x-y plane. The head H also deposits components onto the PCB at a fixed point in the x-y plane. The table T moves in the x-y plane so that components are placed at the desired locations on the PCB. This type of assembly machine is the simplest and is normally adopted when only one type of component needs to be placed.

In a type–2 machine, the feeder (or feeder array) and the table are stationary (and hence the PCB is also held in at a fixed location). The single assembly head H (again, there is no turret) moves in the x-y plane carrying components to the correct placement positions on the PCB. If an array of feeders is adopted, the machine can be employed to place components of different types onto a PCB.

A type–3 machine is the most complex as it has three moving parts:

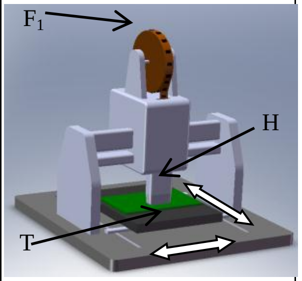
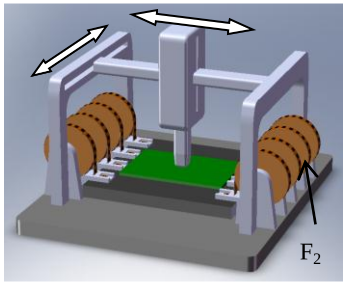
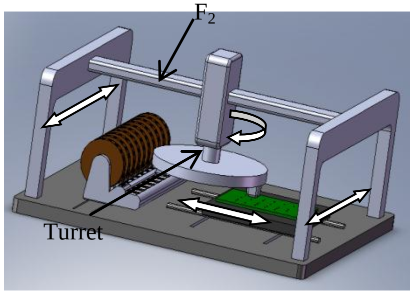
(1) A turret carrying multiple pick-up/placement assembly heads H. The centre of rotation of the turret is fixed in the x-y plane.

(2) A feeder array F_2 that moves along the x-axis to bring the feeder with the required component to the fixed pick-up location.

(3) A table T that moves in the x-y plane to position different points of the PCB, in sequence, at the fixed component placement location.

Component pick-up and placement occur simultaneously after the correct feeder and the table have reached their designated positions and the turret has completed indexing the appropriate pick-up and placement heads (Ayob et al., 2002). This type of machine can also place components of different types. The synchronisation of the multiple feeders, the multi-head turret pick-and-place system and the assembly table is required to perform the task.

Table 5.2 Types of PCB Assembly Machines and their Characteristics (Ang et al., 2009), T: Table, F_1: Single Feeder, F_2: Multiple Feeders, H: Head.

Machine Type	Type–1	Type–2	Type–3
Model of system			
Model of problem	Travelling salesman	Pick and place	Moving Board with Time Delay (MBTD)
Description of assembling process	The machine assembles the components while only the table T that holds the PCB may move in the x-y axis. The component is fed directly into the assembly head H.	The head H moves in the x-y axis to pick and place (assemble) the components while the table T that holds the PCB is stationary. The feeder array F_2 is also stationary.	The multi-head turret H picks the components from the feeder array F_2 (which moves in a single axis to provide the right component for assembly) with one head while placing them with the other onto the PC board after rotating. The PCB is held by a table that moves in the x-y axis in accordance with the sequence and location of the component that needs to be assembled.
Key characteristics of solution	1. The path to assemble the components	1. The feeder slot arrangement. 2. The shortest path to assemble the components.	1. The feeder slot arrangement. 2. The shortest path to assemble the components. 3. The number of heads on the assembly turret.

B. Printed Circuit Board Assembly Planning

PCB assembly planning involves two types of tasks: set-up management and process optimisation. In the context of planning for a single assembly machine with an array of feeders (type–1 machine), set-up management can include arranging the allocation of components among the different feeders which is referred to as the component allocation problem (CAP). In an alternative arrangement, the relative positioning of the feeders in the array requires setting up, this is referred to as the feeder arrangement problem (FAP). Both arrangements are set-up in a manner to reduce assembly cycle times.

Process optimisation for an assembly machine is usually a component sequencing problem (CSP). The aim of component placement sequencing is to optimise the movements of the table (in a type–1 machine) or the assembly head (in a type–2 machine). This problem could be viewed as a typical TSP for which the objective function might be expressed as in Equation 5.1a:

Equation 5.1a

$$D_{Total} = \sum_{i=1}^{N} D(C_i, C_{i+1})$$

(Ho and Ji, 2007b)

Where N is the total number of components to be placed onto a PCB and D(C_i, C_{i+1}) is the distance travelled by the table or the assembly head when moving from the placement position for component C_i to the placement position for component C_{i+1}. (Note that the placement position for C_{N+1} is usually taken to be the same as that for C_1, the starting position, as C_N is the last component to be placed onto a given PCB.)

As the cycle time is the real factor of interest in this problem, the objective function is usually rewritten as in Equation 5.1b:

Equation 5.1b

$$T_{Total} = \frac{D_{Total}}{V}$$

(Ho and Ji, 2007b)

Where V is the average speed of movement of the table or assembly head.

In a type–3 machine, known as the Moving Board with a Time Delay (MBTD), as shown in Figure 5.4, the assembly time is affected by three factors. These are the movement of the PCB, the shifting time of the turret head, and the travelling time of the feeder carrier. The total assembly time needed for a PCB is the summation of the dominating times associated with these three factors for all board components. The formula for the total assembly time is defined in Equation 5.2.

$$T_{Total} = \sum_{i=1}^{N} T_i$$

Equation 5.2

(Ho and Ji, 2007b)

Where N is the total number of components to be placed and T_i, the time required to place the component C_i, is given by Equation 5.3.

$$T_i = \max[t_1(c_{i-1}, c_i), t_2(f_{i+g-1}, f_{i+g}), t_3]$$

Equation 5.3

(Ho and Ji, 2007b)

Where g is the number of assembly heads positioned between the pick-up and placement heads on the turret, t_1 is the time for the table to move from the location of component C_{i-1} to the location of component C_i. The location of C_0 is the starting position of the table, and is given by the Chebyshev metric:

$$t_1(c_i, c_{i-1}) = \max\left(\frac{|x_i - x_{i-1}|}{v_x}, \frac{|y_i - y_{i-1}|}{v_y}\right)$$

Equation 5.4

Ho and Ji, 2007b)

Where x_i = x coordinate of the component $c_{i,}$ and y_i = y coordinate of the component $c_{i,}$, v_x and v_y are the velocities of the x-y table in the x and y directions respectively (both velocities are assigned as 60mm/s for this study),

t_2 is the time for the feeder array to change the pick-up feeder from the feeder supplying component C_{i+g} to that supplying component C_{i+g+1}.

C_{i+g+1} is the component that is to be picked-up when component C_i is placed onto the PCB. (As C_N is the last component to be assembled onto a given PCB, C_{N+g} and C_{N+g+1} are the g^{th} and $(g+1)^{th}$ components to be placed onto the next PCB.) The Chebyshev metric essentially takes into consideration the x-y movement of the board table as independent motions. This is usually the case when the x-y table is controlled by two motors, each separately controlling the x and y movement.

The distance between feeders is measured using the Euclidean metric. The travelling time of the feeder carrier between feeder $f_i(x_i^f, y_i^f)$ and $f_j(x_j^f, y_j^f)$ is given by Equation 5.5.

Equation 5.5

$$t_2(f_i, f_j) = \frac{\sqrt{\left|x_j^f - x_i^f\right|^2 + \left|y_j^f - y_i^f\right|^2}}{v_f}$$

(Ho and Ji, 2007b)

Where x_i is the x coordinate of the feeder $f_{i,}$ y_i is the y coordinate of the feeder f_i, and v_f is the speed of the feeder carrier (v_f is 60mm/s in the case study).

For the case study in this chapter, the feeders are arranged in a straight line (y-axis) as shown in Figure 5.5 and separated by a y-distance of 15 mm. Hence, the *x*-coordinates for the feeders are the same throughout, and any variation due to any possible vibration will be negligible.

t_3 is the time taken by the turret to index the assembly heads by one position and is set as 0.25 seconds per step. Generally, indexing takes place one position at a time and always in the same direction.

As there are three moving parts in this type of machine (a board, a feeder, and a turret), each moving part has to wait for the other two parts to complete their movements before the next component can be picked up or placed. Hence, the time T_i, defined in Equation 5.3, needed for the placement of component i is the maximum time between the board movement t_1, the feeder movement t_2, and the indexing time during each pickup and placement t_3. The problem with this type of machine is that it needs to determine simultaneously the placement sequence of the board components and assign the various types of components to the feeders such that the total assembly time is minimised (Ho and Ji, 2007a).

Another aspect of this process that must be addressed is the more complex, but also more realistic, problem of optimising both the arrangement of feeders denoted as, $F = \{f_1 \ldots, f_j, \ldots, f_{R-1}, f_R\}$, where f_j is the feeder for the j^{th} component type, and the sequence of placement of components denoted as C $=\{c_1\ldots, c_i, \ldots, c_{N\text{-}1}, c_N\}$, where c_i is the i^{th} component to be placed (Figure 5.5) in order to minimise the total assembly time $\mathrm{T}_{\mathrm{Total}}$ as given by Equation 5.2.

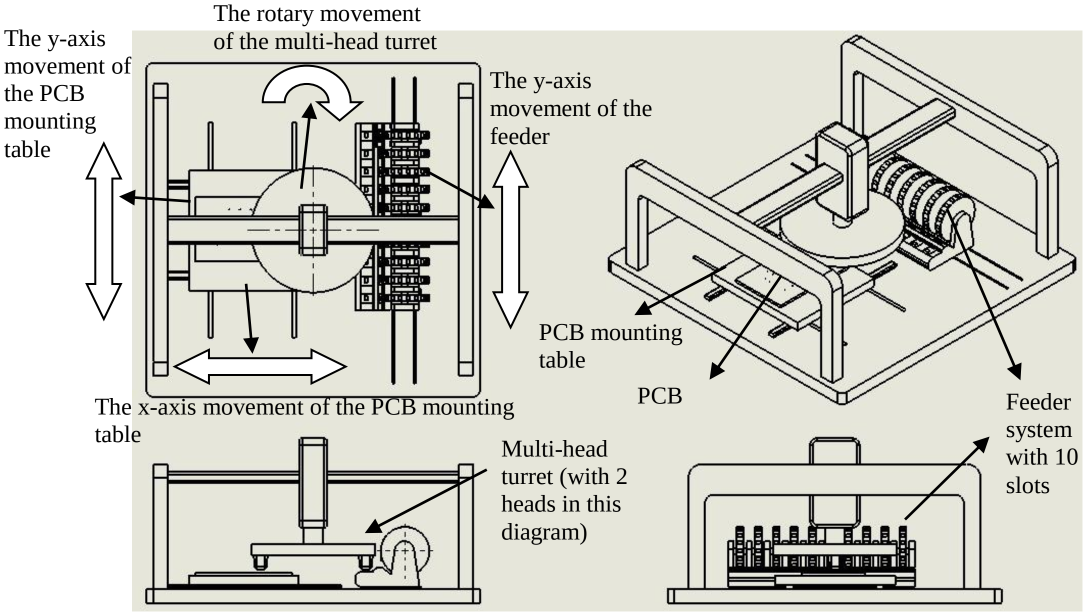

The y-axis movement of the PCB mounting table
The rotary movement of the multi-head turret
The y-axis movement of the feeder
PCB mounting table
PCB
The x-axis movement of the PCB mounting table
Feeder system with 10 slots
Multi-head turret (with 2 heads in this diagram)

Figure 5.4 PCB Assembly Machine of the MBTD Type (with 2 Rotary Turret Heads, 10 Feeder Slots and a Moveable Assembly Table)

Component Assembly Link											**Feeder Arrangement Link**						
c_1	c_2	c_3	c_4	.	.	.	$c_{N\text{-}3}$	$c_{N\text{-}2}$	$c_{N\text{-}1}$	c_N	f_1	f_2	.	.	.	$f_{R\text{-}1}$	f_R

Figure 5.5 Representation of a PCB Assembly Sequence

Previous researchers have applied a variety of techniques to provide a solution to this PCB assembly planning problem. The techniques employed have included the GA (Maimon and Brha, 1998; Wong and Leu, 1993; Ong and Khoo, 1999; Ho and Ji, 2007b), EP (Nelson and Wille, 1995), the minimal spanning tree technique (Leipala and Nevalainen, 1989), and rule-based expert system techniques (Yeo et al., 1996). The GA has been successfully employed to solve the Moving Board with a Time Delay (MBTD) problem (Wong and Leu, 1993; Ong and Khoo, 1999). Four genetic operators and two-links were employed to provide solutions for the component placement and feeder assignment problems. It was shown that the method is easily adaptable to the planning problems of many types of assembly machines. In (Ong and Tan, 2002), the researchers focused on the application of the GA to solve the MBTD problem for a high-speed PCB assembly machine where eight operators (four crossover operators and four mutation operators) were employed. In another paper (Yeo et al., 1996), a rule-based frame system for PCB assembly was developed to generate the component feeder arrangement and placement sequence for concurrent chip placement machines. The system was implemented using an AI programming environment and led to significant time savings in PCB assembly. Ho and Ji (2007b) in their book, focused on the optimisation of the PCB assembly line efficiency. They integrated the component sequencing and the feeder arrangement problems together for the chip shooter machine.

Of these four techniques, the GA and EP seem to be the most popular due to their simple implementation and robustness against local optimum traps where a local, but not global minimum is found. A comparison of the results obtained by other researchers using the GA and EP on a benchmark PCB assembly planning problem will be presented. These results were compared with the results found when applying the Bees Algorithm to the same problem.

5.3.2 The Proposed Bees Algorithm for PCB Benchmark Problems

Figure 5.6 shows the flowchart of the Bees Algorithm when applied to the Component Sequencing Problem (Type–3 PCB machine). After the parameters of the Bees Algorithm have been initialised; the algorithm generates initial solutions (i.e. scout bees) for both component sequencing and feeder arrangement problems. Hence each bee is comprised of two links (as shown in Figure 5.5). The first link represents the sequence of component placements and is generated as a simple TSP. The second link representing the feeder arrangement is generated randomly.

The scout bees are normally created by randomly generating feeder arrangements and placement sequences and then checking that they are valid. A valid feeder arrangement would be a permutation of feeder labels, each corresponding to a particular feeder and component type. For example, label (A) might correspond to feeder (A) which supplies 100KΩ resistors. The number of labels in a valid feeder arrangement would be equal to the number of feeders in the assembly machine. A valid placement sequence would also be a permutation of labels. Each label representing a placement position on the PCB, with the total number of labels (i.e. the length of the sequence) being equal to the number of placement positions.

Next, the fitness of each bee is measured, where each fitness is given as the assembly time, found using Equation 5.3. The fitness values are then sorted and the bees with higher fitness values are selected and a neighbourhood search is initiated using the 2-Opt local search heuristic for the first link and the single-point insertion algorithm for the second link. The fittest bee from each site is then selected to form the next bee population. The remaining bees in the population are placed randomly around the search space to scout for new potential solutions (sequences). This process is repeated until either the best fitness value has been found or the specified maximum number of iterations has been reached.

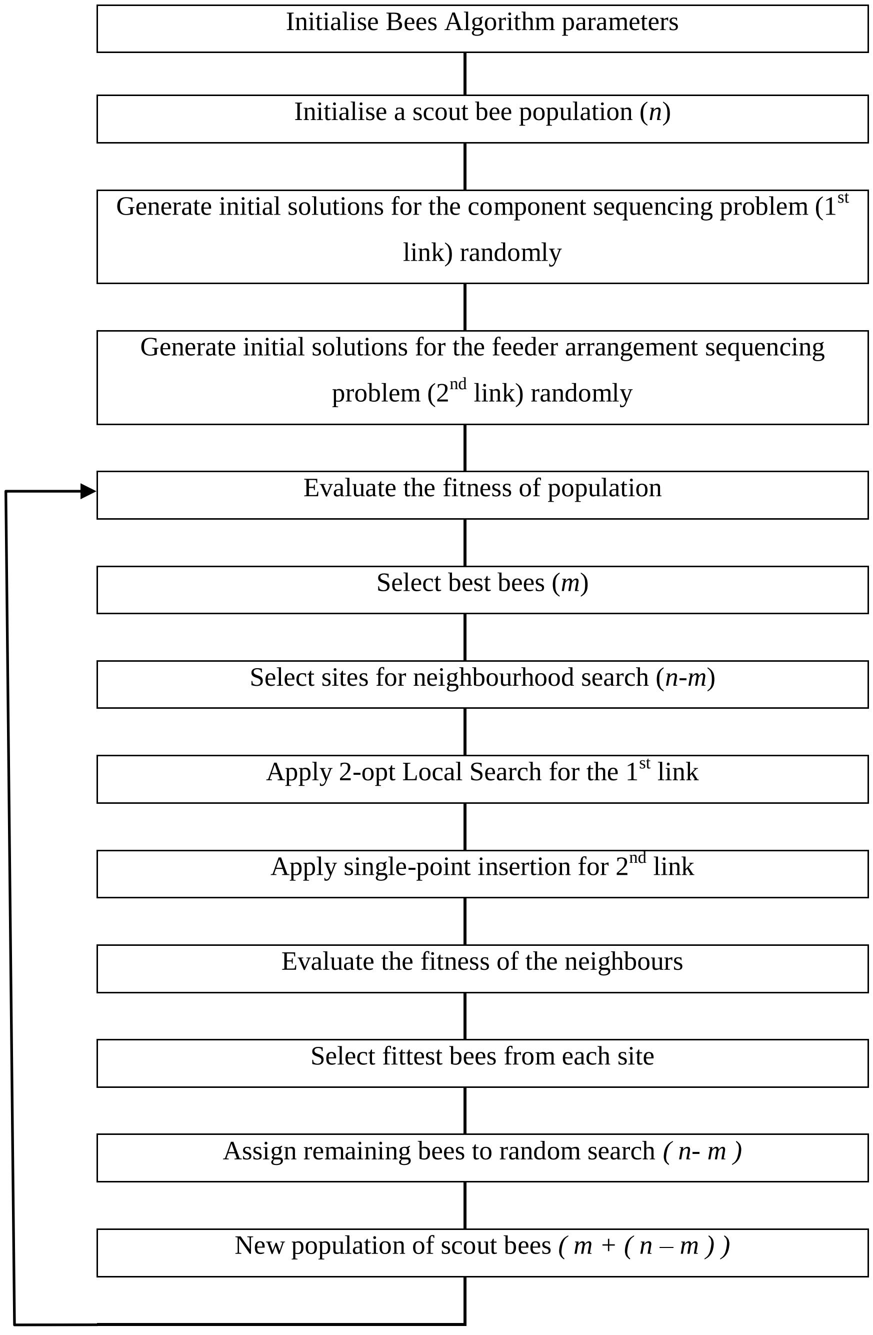

Initialise Bees Algorithm parameters
Initialise a scout bee population (*n*)
Generate initial solutions for the component sequencing problem (1st link) randomly
Generate initial solutions for the feeder arrangement sequencing problem (2nd link) randomly
Evaluate the fitness of population
Select best bees (*m*)
Select sites for neighbourhood search (*n*-*m*)
Apply 2-opt Local Search for the 1st link
Apply single-point insertion for 2nd link
Evaluate the fitness of the neighbours
Select fittest bees from each site
Assign remaining bees to random search *(n- m)*
New population of scout bees *(m + (n – m))*

Figure 5.6 Flowchart of Bees Algorithm Solving a PCB Problem

5.3.3 PCB Component Assembly Using a MBTD Machine

The problem employed to test the ability of the BA to perform PCB assembly planning for a type–3 machine is detailed in (Leu et al., 1993). For this problem, the number of components to be placed onto a PCB is 50 and there are ten feeders, each supplying a different type of component. The coordinates of the placement positions and the parameters of the assembly machine (speed of movement of the table, assembly heads and indexing time of the turret) summarised in Table 5.3, are also given in (Leu et al., 1993).

At first, the PCB problem was solved with the initial positions of the scout bees chosen completely randomly, as described above. Further improvement to the Bees Algorithm was then made by introducing a good patch (a good initial solution) to one of the scout bees. The location of that patch was found by optimising only the component placement sequence (the first link) thus treating the problem as a simple TSP.

Table 5.3 The Parameters of the MBTD Assembly Machine

Number of components	50
Number of feeders	10
Number of turret heads	2
Indexing time of turret	0.25s/index
Average PCB mounting table speed	60mm/s
Average feeder system speed	60mm/s
Distance between feeders	15mm

5.3.4 Experimental Results

The main features of the Bees Algorithm employed in this work have been chosen by running a set of trail experiments and shown in Table 5.4. As can be noticed from this table, the number of parameters in the Bees Algorithm has been reduced from six to three. This is because the elite sites parameter (*e*) was not employed and hence the number of recruited bees (*nep*) around it was not employed either. In addition, the patch size parameter (*ngh*) was not needed because a local search operator e.g. either the 2-Opt or single-point insertion, was employed for this type of combinatorial problem.

Table 5.4 also shows that the population size *n* was initialised to 100 bees for the first iteration. This then dropped to be equal to the number of selected sites *m* (20 sites). The recruited bees were then: *nep,* that is 50 bees for each of the best *m* sites.

5.4 DISCUSSION

Figure 5.7 shows the evolution of the best assembly time as the operation of the Bees Algorithm progresses for the first case where a good starting solution was not chosen (without “seeding”). After 160 iterations, the best assembly time was found to be 25.92 seconds. The corresponding component placement sequence is shown in Figure 5.8.

For the second case where seeding was employed, the sequence produced an assembly time of 29 seconds as shown in Figure 5.9.

Figure 5.10 shows the evolution of the best assembly time and Figure 5.11 the optimal sequence after 100 iterations with an assembly time of 24.08 seconds.

For comparison, Table 5.5 presents the above results found using the Bees Algorithm alongside those produced by other researchers for the same problem using the GA (Ho and Ji, 2007b; Leu et al., 1993; Ong and Tan, 2002) and EP (Nelson and Wille, 1995).

According to Table 5.5, the performance of the Bees Algorithm with seeding (24.08 seconds) is superior to that of the simple Bees Algorithm (25.92 seconds), the simple GA (51.5 seconds), the hybrid GA (25.5 second) and EP (36 seconds) when applied to a benchmark assembly task. In addition, the number of evaluations needed to perform the optimisation process using the Bees Algorithm with seeding is less than any of the others.

Table 5.4 The Parameters of the Bees Algorithm

Bees Algorithm parameters	Symbol	Value	
		Completely random initial population	Seeding initial population with a good solution
Population size	*n*	100 (1st iteration only)	100 (1st iteration only)
		n = *m* (for iteration 2 onwards)	*n* = *m* (for iteration 2 onwards)
Number of selected sites	*m*	20	20
Number of recruited bees for best m sites	*nep*	50	50
Number of iterations	*itr*	160	100

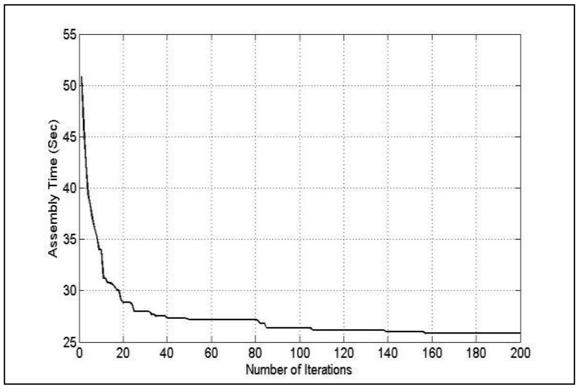


Figure 5.7 Evolution of Best Assembly Time for Case 1 – Without Seeding

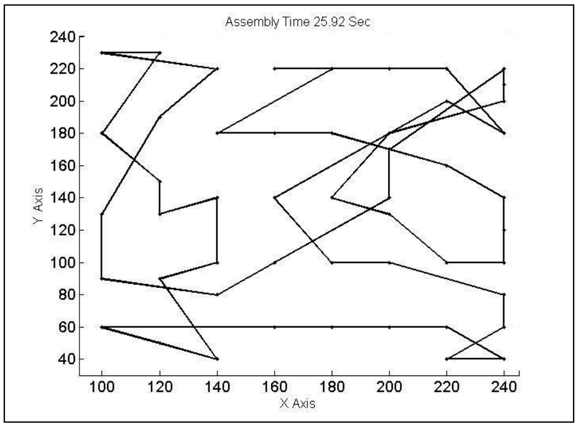


Figure 5.8 Optimal Assembly Sequence (25.92 sec) after 160 Iterations for case 1 – Without Seeding

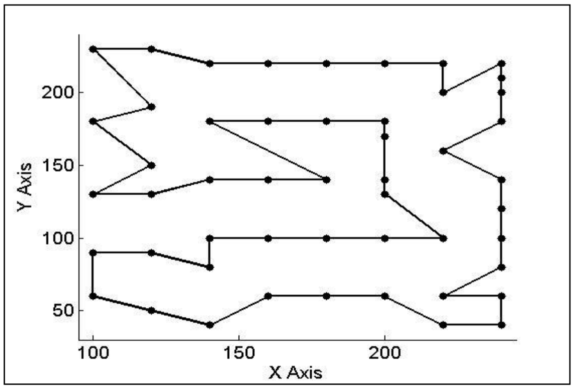


Figure 5.9 Assembly Sequence (29.00 sec) Employed as a Seed for Case 2 – With Seeding

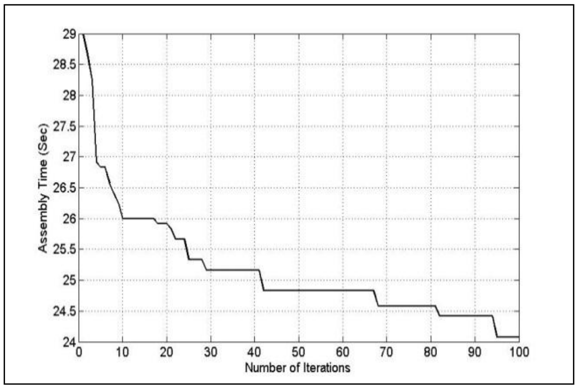


Figure 5.10 Evolution of Best Assembly Time (29.00 sec) for case 2 – With Seeding

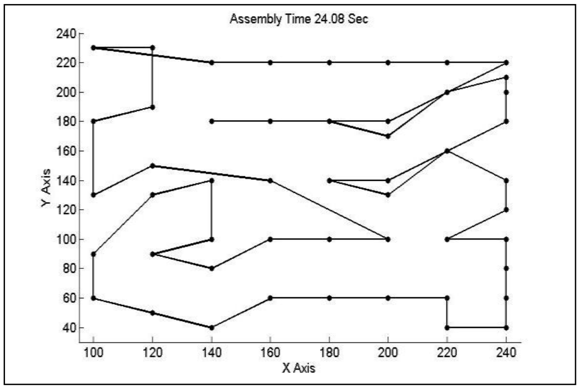


Figure 5.11 Optimal Assembly Sequence (24.08 sec) after 100 Iterations for case 2 – With Seeding

Table 5.5 Results Obtained using Different Algorithms

References	**1**	**2**	**3**	**4**	**Bees Algorithm**	
Optimisation technique	**GA**	**EP**	**GA**	**HGA**	**No seeding**	**With seeding**
Best initial solution (sec)	70	n/a	60	28.83	54.59	**29**
Number of evaluations	175,000	500,000	50,000	**14,150**	160,000	100,000
Best assembly time (sec)	51.5	36	26.9	25.5	25.92	**24.08**
1: (Leu et al., 1993) **2**: (Nelson and Wille, 1995)			**3**: (Ong and Tan, 2002) **4**: (Ho et al., 2007b)			

5.5 SUMMARY

This chapter has described the application of the Bees Algorithm to the optimisation of pick-up and placement sequences for a PCB assembly machine. For a benchmark assembly problem, the optimal assembly time of 24.08 seconds obtained by the Bees Algorithm was 6% shorter than the best assembly time reported in the literature to date, 25.5 seconds. If the assembly machine was employed continuously, 24 hours a day, 365 days a year, such a saving in assembly time would represent an annual increase in production of some 72,000 PCBs.

The performance of the Bees Algorithm for solving the component sequencing and feeder arrangement problems introduced, showed a significant improvement in the assembly time. However, this reduction in assembly time achieved by the Bees Algorithm was at the expense of a high number of candidate solutions that had to be generated and evaluated (as seen with the number of evaluations presented in Table 5.5).

The computational experiments presented in the chapter have demonstrated that the Bees Algorithm provides a significant reduction in assembly time compared to the results obtained using the GA and EP when applied to a benchmark assembly task.

CHAPTER 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 the Bees Algorithm as a swarm-based tool for solving complex optimisation problems.

The specific contributions were:

- Employing the Bees Algorithm for tuning its own parameters;

- Using the Bees Algorithm for stochastic optimisation problems;

- Explaining how the Bees Algorithm can be efficiently applied to combinatorial optimisation problems, namely, the Travelling Salesman and Printed Circuit Board problems;

- Reducing the number of parameters in the Bees Algorithm from six to three. This was attained by making some of the parameters dependent on others and by the selection of the best sites only and not selecting the elite bees as is the case with the standard Bees Algorithm. This in return eliminated the need to choose the number of recruited bees around the elite sites (*nep)*. In addition, the patch size parameter was not required because a local search operator e.g. either the 2-Opt or single-point insertion, was employed for this type of combinatorial problem.

6.2 CONCLUSIONS

All algorithms have a set of parameters that control their behaviour. These are employed in order to improve the performance of the algorithm when trying to arrive at solutions to the problem at hand. This set of parameters depends on the problem being tackled and its dimension. Unfortunately, these parameters cannot be generalised to any type of problem of any size.

To arrive at a suitable set of parameters, different algorithms employ different techniques.

The use of the activity of foraging bees as model for an optimisation process was shown to provide substantial benefits especially when compared with other optimisation techniques.

The research in this thesis has extended the application of the Bees Algorithm to the solution of complex optimisation problems. The new modified Bees

Algorithm was employed to optimise stochastic problems based on statistical analysis. The number of parameters of a standard Bees Algorithm was reduced from six to three which makes the tuning easier. The algorithm can also adapt itself to the problem it is dealing with.

When combined with a local search algorithm (2-Opt), the Bees Algorithm was also able to handle combinatorial problems.

6.3 FUTURE WORK

There are a number of directions that can be pursued in order to enhance the Bees Algorithm and widen its application potential.

1. The Bees Algorithm could be combined with any of the Ant Colony Optimisation Algorithms such as Ant Colony System (ACS) to ensure higher performance and solution quality. For example, a new algorithm could be developed that employs the Bees Algorithm as a global search and the ACS as a local search.

2. The Bees Algorithm was employed to tune its own parameters. It can be tested on the tuning of other algorithms and techniques, for example, online tuning of the Particle Swarm Optimisation (PSO) algorithm.

3. The Bees Algorithm could be applied to other types of combinatorial optimisation problems, such as Vehicle Routing, Job-Shop Scheduling and Quadratic Assignment Problems and to other types of assembly machine such as Pick and Place Machines (PAPs).

4. More statistical analysis could be added to the modified Bees Algorithm. For example, the standard deviation of the fitness values which indicates the stability of the solution, a smaller standard deviation meaning a more stable solution, could be added as part of a multi-objective function to the modified Bees Algorithm to find the best and most stable solution.

REFERENCES

Aarts, E. H. L., Korst, J. H. M. and Vanlaarhoven, P. J. M. 1988. A quantitative analysis of the simulated annealing algorithm – A case study for the traveling salesman problem. *Journal of Statistical Physics*, 50(1-2), pp. 187-206.

Adenso-Diaz, B. and Laguna, M. 2006. Fine-tuning of algorithms using fractional experimental designs and local search, *Operations Research*, 54(1): pp. 99-114.

Ang, M.C., Pham, D.T., Ng, K.W. 2009. Application of the Bees Algorithm with TRIZ-inspired operators for PCB assembly planning. In: *Proceedings of 5th Virtual International Conference on Intelligent Production Machines and Systems (IPROMS2006),* Cardiff, UK, 2009. pp. 454-459.

Ayob, M., Cowling, P. and Kendall, G. 2002. Optimisation for surface mount placement machines. In: *Proceeding of IEEE ICIT'02*. Bangkok, 2002, pp. 498-503.

Azeem, M.F. and Saad A.M. 2004. Modified Queen Bee Evolution Based Genetic Algorithm for Tuning of Scaling Factors of Fuzzy Knowledge Base Controller. IEEE INDICON 2004 Proceedings of the India Annual Conference, 299-303.

Bahamish, H. A. A., Abdullah, R. and Abdul Salam, R. A. 2008. Protein Conformational Search Using Bees Algorithm. In: *2nd IEEE Asia International Conference on Modeling and Simulation, AICMS 08*. Kuala Lumpur, Malaysia: IEEE Computer Society. pp. 911-916.

Baeck, T., Hoffmeister, F. and Schwefel, H.P. 1991. A survey of evolution strategies. In: *Proceedings of 4th International Conference on Genetic Algorithm.* San Mateo - USA, 1991. Morgan Kaufmann, pp. 2-9.

Bartz-Beielstein, T. and Markon, S. 2004. Tuning search algorithms for real-world applications: A regression tree based approach. In: *Proceedings of Evolutionary Computation*, Portland, OR, USA, 2004, pp. 1111-1118.

Beekman, M., Fathke, R., and Seeley, T. 2006. How does an informed minority of scouts guide a honey bee swarm as it flies to its new home? *Animal Behavior*, 71(1):161–171.

Bilchev, G. and Parmee, I.C. 1995. The ant colony metaphor for searching continuous design spaces, in *Selected Papers from AISB Workshop on Evolutionary Computing*. Springer-Verlag: London. p. 25-39.

Birattari, M., Sützle, T., Paquete, L. and Varrentrapp, K. 2002 A racing algorithm for configuring metaheuristic. In*: Proceedings of Genetic and Evolutionary Computation Conference.* New York, USA, 2002, pp. 11-18.

Blackwell, T. and Branke, J., 2004. *Multi-swarm optimization in dynamic environments, Applications of Evolutionary Computing,* in *Lecture Notes in Computer Science,* Raid,l G.R., Editor, Springer-Verlag: Berlin, Germany.

Bonabeau, E., 1998. Social insect colonies as complex adaptive systems, *Ecosystems*, (1), pp. 437-443.

Bonabeau, E., Dorigo, M., and Theraulaz, G. 1999. *Swarm intelligence: from natural to artificial systems.* New York: Oxford University Press.

Burke, E.K., Cowling, P.I. and Keuthen, R. 1999. New models and heuristics for component placement in printed circuit board assembly. In: *Proceedings of ICIIS9*, 1999, pp. 133-140.

Camazine, S., Deneubourg, J.L., Franks, N.R., Sneyd, J., Theraula, G., and Bonabeau, E. 2003. *Self-organization in biological systems.* Princeton University Press.

Chandra, B., Karloff, H. and Tovey, C. 1999. New results on the old k-opt algorithm for the traveling salesman problem. *SIAM Journal on Computing*. 28(6), pp. 1998-2029.

Coy, S.P., Golden, B.L., Runger, G.C., and Wasil, E.A. 2001. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7 (1): pp. 77-97.

Deneubourg, J.L., Aron, S., Goss, S. and Pasteels, J.M., 1990. The self-organising exploratory pattern of the Argentine ant. *Journal of Insect Behaviour* (3), pp. 159-168.

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

Dorigo, M. and Stützle, T. 2004. *Ant colony optimization*, Cambridge: MIT Press.

Dorigo, M., Di Caro, G., and Gambardella, L.M. 1999. Ant algorithms for discrete optimization. *Artificial Life*, 5(2), pp. 137-172.

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

Dréo, J. and Siarry, P., 2004. Continuous interacting ant colony algorithm based on dense heterarchy, *Future Generation Computer Systems*, (20), pp. 841–856.

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

Engelbrecht, A. P. 2005. *Fundamentals of computational swarm intelligence.*

Hoboken, N.J. Wiley.

Farooq, M 2008. Bee-Inspired Protocol Engineering: From Nature to Networks, Natural Computing Series, Springer; London, UK.

Figlali, N., Özkale, C., Engin, O. and Figlali, A. 2005. Investigation of ant system parameter interactions by using design of experiments for job-shop scheduling problems. In: *35th International Conference on Computers and Industrial Engineering*, Istanbul, Turkey, 2005, pp. 745-750.

Freisleben, B. and Merz, P. 1996. A genetic local search algorithm for solving symmetric and asymmetric travelling salesman problems. In: *Proceedings of International Conference on Evolutionary Computation*. Indianapolis, USA, 1996, pp. 616-621.

Frisch, K.V. 1976. *Bees: their vision, chemical senses and language*. Revised Edition ed. Ithaca, N.Y., Cornell University Press.

Fogel, D.B. 2000. *Evolutionary computation: toward a new philosophy of machine intelligence*, 2nd edition, New York: IEEE Press.

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

Gaertner, D. and Clark, K. 2005. On optimal parameters for ant colony optimization algorithm. In: *Proceedings of the International Conference on Artificial Intelligence*, Las Vegas, Nevada, USA, CSREA, 2005, pp. 83 - 89.

Gambardella, L.M. and Dorigo, M. 1996. Solving symmetric and asymmetric TSPs by ant colonies. In: *Proceedings of IEEE International Conference on Evolutionary Computation.* Nayoya University, Japan, 1996, pp. 622-627.

Garey, M.R. and Johnson, D.S. 1979. *Computers and intractability: A guide to the theory of NP completeness*. San Francisco: Freeman.

Ghanbarzadeh, A. 2007. *The Bees algorithm. A novel optimisation tool.* PhD. Cardiff University.

Glover, F. and Kochenberger, G. A. 2003. *Handbook of metaheuristics,* Boston, MA, USA: Kluwer.

Goldberg, D.E. 1989. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley Longman Publishing Co., MA.

Haddad, O.B., Afshar, A. and Marino, M.A., 2006. Honeybees mating optimization (HBMO) algorithm: A new heuristic approach for water resources optimization, *Water Resources Management*, (20), pp. 661–680.

Hao, Z.-F., Cai, R.-C., and Huang, H. 2006. An adaptive parameter control strategy for ACO. In: *Proceedings of International Conference on Machine Learning and Cybernetics,* Dalian, China, 2006, pp. 203-206.

Helsgaun, K, 2000. An effective implementation of the Lin-Kernighan travelling salesman heuristic. *European Journal of Operational Research* 126(1), pp. 106-130.

Ho, W. and Ji, P. 2007a. A genetic algorithm to optimise the component placement process in PCB assembly. *International Journal of Advanced Manufacturing Technology* 26, pp. 1397–1401.

Ho, W. and Ji, P. 2007b. *Optimal production planning for PCB assembly*. London: Springer.

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

Ibaraki, T. 1997. *Combination with other optimization methods. Handbook of evolutionary computation*. Ed. Back, T., Fogel, D.B. and Michalewicz, Z. IOP Publishing Ltd and Oxford University Press.

Jain, A. K. and Dubes, R. C. 1988. Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs, NJ.

Johnson, D.S. and McGeoch, L.A. 1997. *Local search in combinatorial optimization*, Ed. Aarts, E. and Lenstra J.K. Wiley, Chichester, p. 215.

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

Kennedy, J. and Eberhart, R. 1995. Particle swarm optimization, In: *Proceedings of 1995 IEEE International Conference on Neural Networks*, Perth - Western Australia, 4, 1995, pp. 1942-1948.

Kohonen, T. 1989. Self-Organising and Associative Memory (3rd ed.). Springer-Verlag ,Berlin.

Kohonen, T. 1990. The Self-Organising Map". Procs. IEEE, 78(9), pp. 1464-1480.

Knox, J., 1994. Tabu Search Performance on the Symmetric Traveling Salesman Problem, *Computers Ops Res*., 21(8), pp. 867-876.

Laporte, G. 1992. The travelling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59(2). pp. 231-247.

Lara, C., Flores, J. J. and Calderón, F. 2008. Solving a School Timetabling Problem Using a Bee Algorithm. Lecture Notes in Computer Science, 2008, 5317, pp. 664-674.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. 1985. *The traveling salesman problem: a guided tour of combinatorial optimization*. John Wiley and Sons, NY.

Lee, J. Y. and Haj Darwish, A. 2008. Multi-objective Environmental/Economic

Dispatch Using the Bees Algorithm with Weighted Sum. In: EU-Korea Conference on Science and Technology (EKC2008). Heidelberg, Germany: Springer. pp. 267-274.

Leipala, T. and Nevalainen, O. 1989. Optimisation of the movements of a component placement machine. *European Journal of Operational Research*. 38, pp. 167-177.

Leu, M.C., Wong, H. and Ji, Z. 1993. Planning of component placement/insertion sequence and feeder setup in PCB assembly using genetic algorithm*.* ASME. *Trans. Journal of Electronic Packaging.* 115, pp. 424-432.

Lin, S. and Kernighan, B.W. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*. 21(2), pp. 498-516.

Ling, W. and Luo, H. 2007. An adaptive parameter control strategy for ant colony optimization. In: *Proceedings of International Conference on Computational Intelligence and Security*. Harbin - China, 2007, pp. 142-146.

Lucic, P. and Teodorovic, D. 2003. Computing with bees: attacking complex transportation engineering problems. *International Journal on Artificial Intelligence Tools*. 12(3), pp. 375-394.

Maimon, O.Z. and Brha, D. 1998. A genetic algorithm approach to scheduling PCBs on a single machine. *International Journal of Production Research* 36(3): pp. 761- 784.

Martens, D., De Backer, M., Haesen, R., Vanthienen, J., Snoeck, M., Baesens, B. 2007. Classification with Ant Colony Optimization", *IEEE Transactions on Evolutionary Computation*, 11(5), pp. 651-665.

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), pp. 3814-3822.

Merz, P. and Freisleben, B. 1997. Genetic local search for the TSP: New results. In: *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*. Indianapolis, USA, 1997, pp. 159-164.

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

Nelson, K.M. and Wille, L.T. 1995. Comparative study of heuristics for optimal printed circuit board assembly. In: *Proceedings of Southcon 95.* Fort Lauderdale, FL, USA, 1995, pp. 322 - 327.

Okano, H., Misono, S. and Iwano, K. 1999. TSP construction heuristics and their relationships to the 2-Opt. *Journal of Heuristics* (1), pp. 71-88, Kluwer Academic Publishers, Boston.

Oliver, I. M., Smith, D.J. and Holland, J.R.C. 1987. A study of permutation crossover operators on the traveling salesman problem. In: *Proceedings of Second Int. Conf. Genetic Algorithms and their Applications*. Cambridge, MA, USA, 1987, pp. 224-230.

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.

Ong, N. and Khoo, L.P. 1999. Genetic algorithm approach in PCB assembly. *Integrated Manufacturing Systems,* 10(5): pp. 256-265.

Ong, N.S. and Tan, W.C. 2002. Sequence placement planning for high-speed PCB assembly machine. *Integrated Manufacturing Systems*. 13 (1), pp. 35-46.

Passino, K.M., 2002. Biomimicry of bacterial foraging for distributed optimisation and control, *IEEE Control Systems Magazine* (June), pp. 52-67.

Passino, K.M., Seeley, T.D., and Visscher, P.K., 2008. Swarm cognition in honeybees, *Behavioral Ecology Sociobiology*, 62(3), pp. 401-414.

Pellegrini P., Favaretto D. and Moretti E. 2006. *On MAX -MIN ant system's parameters*, in *Ant Colony Optimization and Swarm Intelligence*. pp. 203-214.

Pham, D.T., Afify, A.A. and Koç, E. 2007b. Manufacturing cell formation using the Bees Algorithm. In: *3rd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2007)*, Whittles, Dunbeath, Scotland, 2007, pp. 523-528.

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

Pham, D.T. and Sholedolu, M. 2008. Using a hybrid PSO-Bees Algorithm to train Neural Networks for Wood Defect Classification. In: *4th International Virtual Conference on Intelligent Production Machines and Systems, IPROMS.*

Pham, D.T., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S. and Zaidi, M. 2006b. The Bees Algorithm - A novel tool for complex optimisation problems. In: *Proceedings of 2nd Virtual International Conference on Intelligent Production Machines and Systems (IPROMS2006)*, Cardiff, UK, 2006. pp. 454-459.

Pham, D.T., Koc, E., Ghanbarzadeh, A., and Otri, S. 2006c. Optimisation of the weights of multi-layered perceptrons using the Bees Algorithm, in *5th International Symposium on Intelligent Manufacturing Systems*, Sakarya, Turkey, 2006.

Pham, D. T., Koc, E., Lee, J.Y. and Phrueksanant, J., 2007c. Using the Bees Algorithm to schedule jobs for a machine. In: *LAMDAMAP, 8th international Conference on Laser Metrology, CMM and Machine Tool Performance*, Cardiff, Euspen, UK, 2007, pp. 430-439.

Pham D. T., Lee J. Y., Haj Darwish A., and Soroka A. J. 2008a. Multi-objective Environmental/Economic Power Dispatch using the Bees Algorithm with Pareto optimality. In: *4th International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2008a),* Whittles, Dunbeath, Scotland, 2008.

Pham, D.T., Castellani, M. and Fahmy, A.A., 2008e. Learning the inverse kinematics of a robot manipulator using the Bees Algorithm. In: *Proceedings of INDIN*, 2008, pp. 493-498.

Pham, D. T., Castellani, M., Sholedolu, M. and Ghanbarzadeh, A. 2008c. The Bees Algorithm and Mechanical Design Optimisation. pp250-255

Pham, D.T., and Karaboga, D., 2000. *Intelligent optimisation techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. Springer-Verlag, London, UK.

Pham, D.T. and Castellani, M. 2009. The Bees Algorithm – Modelling Foraging Behaviour to Solve Continuous Optimisation Problems. *Proc. ImechE, Part C*, 223(12): 2919-2938.

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

Pham D. T. and Sahran S., 2006. "Control Chart Pattern Recognition with Spiking Neural Networks", *Proceedings of the Second Virtual International Conference on Intelligent Production Machines and Systems*, D. T. Pham, E. E. Eldukhri and A. J. Soroka (eds), Elsevier (Oxford), 319-325, ISBN 0-08-045157-8.

Pham, D.T., Castellani, M. and Ghanbarzadeh, A., 2007e. Preliminary design using the Bees Algorithm. In: *Proceedings of Eighth LAMDAMAP International Conference on Laser Metrology*, CMM and Machine Tool Performance, Cardiff, UK, 2007, pp. 420-429.

Pham, D.T., Ghanbarzadeh, A., Koc, E., and Otri. S., 2006d. Application of the Bees Algorithm to the training of radial basis function networks for control chart pattern recognition. In: *5th CIRP International Seminar on Intelligent Computation in Manufacturing Engineering, CIRP ICME,* 2006, Ischia, Italy.

Pham, D.T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M., 2005. The Bees Algorithm. Technical Note, Manufacturing Engineering Centre, Cardiff University, UK.

Pham, D. T., Muhamad, Z., Mahmuddin, M., Ghanbarzadeh, A., Koc, E. and Otri, S. 2007a. Using the Bees Algorithm to optimise a support vector machine for wood defect classification. IPROMS 2007 Innovative Production Machines and Systems Virtual Conference, Cardiff, UK.

Pham, D. T., Pham, Q. T., Ghanbarzadeh, A., and Castellani, M. 2008b. Dynamic Optimisation of Chemical Engineering Processes Using the Bees Algorithm. In: *17th IFAC World Congress COEX*, South Korea, 2008, pp. 6100-6105.

Pham, D.T., Otri, S. and Darwish, A.H., 2007d. Application of the Bees Algorithm to PCB assembly optimisation. In: *IPROMS, 3rd International Virtual Conference on Intelligent Production Machines and Systems.* Whittles, Dunbeath, Scotland. 2007, pp. 511-516.

Pham, D.T., Soroka, A.J., Ghanbarzadeh, A., Koc, E., Otri, S. and Packianather, M.,2006e. Optimising neural networks for identification of wood defects using the Bees Algorithm. In: *IEEE International Conference on Industrial Informatics*, Singapore, 2006, pp. 1346-1351.

Pham, D.T., Otri, S., Afify, A., Mahmuddin, M., and Al-Jabbouli, H. 2007f. Data clustering using the Bees Algorithm. In: *40th CIRP International Manufacturing Systems Seminar* Liverpool, UK, 2007.

Pham, D. T., Otri, S., Ghanbarzadeh, A., and Koc, E. 2006a. Application of the Bees Algorithm to the Training of Learning Vector Quantisation Networks for Control Chart Pattern Recognition," in *2nd IEEE International Conference on Information and Communication Technologies: From Theory to Applications*. Damascus, Syria, 2006, (1), pp. 1624 -1629.

Pilat, M.L. and White, T. 2002. Using Genetic Algorithms to optimize ACS-TSP. In: *Proceedings of the Third International Workshop on Ant Algorithms.* Springer-Verlag, 2002, pp. 282-287.

Rechenberg, I. 1965. Cybernetic solution path of an experimental problem, *Library Translation no. 1122.* Ministry of Aviation, Royal Aircraft Establishment, Farnborough, Hants UK.

Rego, C. and Glover, F. 2002. *Local search and metaheuristics*. Gutin and Punnen, pp. 309 - 368.

Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M. 1977. An analysis of several heuristics for the travelling salesman problem. *SIAM Journal on Computing* 6(2), pp. 563-581.

Saad, E.M., Awadalla, M.H., Darwish, R.R. 2008"A Data Gathering Algorithm for a Mobile Sink in Large-Scale Sensor Networks," *Wireless and Mobile Communications, 2008. ICWMC '08. The Fourth International Conference on*, vol., no., pp.207-213.

Sadik, S., Ali, A., Ahmad, F. and Suguri, H. 2006. "Using Honey Bee Teamwork Strategy in Software Agents," *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, pp.1-6.

Sato, T., and Hagiwara, M. 1997. Bee System: finding solution by a concentrated search. In: *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, USA, 1997, pp. 3954-3959.

Seeley, T.D. 1996. *The wisdom of the hive: The social physiology of honey bee colonies*, Cambridge, Massachusetts: Harvard University Press.

Seeley, T. D. and Visscher, P. K. 2003. Choosing a home: how the scouts in a honey bee swarm perceive the completion of their group decision making. Behav Ecol Sociobiol, (54) pp. 511–520.

Socha K. 2003. The influence of run-time limits on choosing ant system parameters. In: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*. Chicago, USA, 2003 (2723), pp. 49-60.

Solnon, C. 2002. Boosting ACO with a preprocessing step. Applications of Evolutionary Computing, Kinsale, Ireland, pp. 41-50.

Stützle, T., 1997. MAX-MIN Ant System for the quadratic assignment problem. Technical Report, AIDA-97-4, FB Informatik, TU Darmstadt, Germany, 1997.

Stützle, T. and Hoos, H. 1997. MAX-MIN ant system and local search for the traveling salesman problem. In: *Proceedings of ICEC'97 - 1997 IEEE 4th International Conference on Evolutionary Computation*. Indianapolis, USA, 1997, pp. 308-313.

Sung, H. J. 2003. *Queen-Bee Evolution for Genetic Algorithms*. Electronic Letters, 39(6), 575- 576.

Teodorovic, D., Lucic, P., Markovic, G. and Orco, M. D. 2006. Bee colony optimization: principles and Applications, Neural Network Applications in Electrical Engineering, pp. 151–156.

Tereshko, V. and Lee, T., 2002. How information-mapping patterns determine foraging behaviour of a honey bee colony, *Open Systems and Information Dynamics*, 9(2), pp. 181-193.

Tovey, C. A. 2004. The honey bee algorithm, a biologically inspired approach to internet server optimization. Engineering Enterprise. pp.13-15.

Watanabe, I. and Matsui, S. 2003. Improving the performance of ACO algorithms by adaptive control of candidate set. In: *Proceedings of Evolutionary Computation Conference*, Chicago, IL, USA, 2003 (2), pp. 1355-1362.

Wedde, H.F., Farooq, M. and Zhang Y. 2004. *BeeHive: An efficient fault-tolerant routing algorithm inspired by honey bee behavior*. 2nd ed. Lecture Notes in Computer Science. 83-94.

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 behaviour. In: *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*. ACM Press: Washington DC, USA, 2005, p. 153-160.

White, C.M. and Yen, G.G. 2004. A hybrid evolutionary algorithm for travelling salesman problem. In: *Proceedings of Congress on Evolutionary Computation, CEC,* 2004. pp. 1473-1478.

Wiener, R. 2003. Branch and bound implementations for the travelling salesperson problem - Part 1. *Journal of Object Technology* 2(2), pp. 65-86.

Wolpert, D.H. and Macready, W.G., 1997. No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation*, 1(1), pp.:67-82.

Wong, H. and Leu, M.C. 1993. Adaptive genetic algorithm for optimal printed circuit board assembly planning. *Annals of the CIRP*. 42, pp. 17-20.

Wong, K. Y., Komarudin, 2008. Parameter tuning for ant colony optimization: A review. In: *Proceedings of the International Conference on Computer and Communication Engineering*. Kuala Lumpur - Malaysia, 2008. pp. 542-545.

Yang, X.S. 2005. Engineering optimizations via nature-Inspired virtual Bee Algorithms. In: *IWINAC (2)*. Springer. p. 317-323.

Yeo, S.H., Low, C.W. and Yong, K.H. 1996. A rule-based frame system for concurrent assembly machines. *The International Journal of Advanced Manufacturing Technology*. 12, pp. 370-376.

Zecchin, A.C., Simpson, A.R., Maier, H.R. and Nixon, J.B. 2005. Parametric study for an ant algorithm applied to water distribution system optimization. *IEEE Transactions on Evolutionary Computation*. 9(2): pp. 175-191.