MACHINE SCHEDULING USING

THE BEES ALGORITHM

A thesis submitted to Cardiff University

for the degree of

Doctor of Philosophy

by

JANYARAT PHRUEKSANANT

Mechanics, Materials and Advanced Manufacturing

School of Engineering

Cardiff University

United Kingdom

July 2013

ABSTRACT

Single-machine scheduling is the process of assigning a group of jobs to a machine. The jobs are arranged so that a performance measure, such as the total processing time or the due date, may be optimised. Various swarm intelligence techniques as well as other heuristic approaches have been developed for machine scheduling. Previously, the Bees Algorithm, a heuristic optimisation procedure that mimics honeybee foraging, was successfully employed to solve many problems in continuous domains. In this thesis, the Bees Algorithm is presented to solve various single-machine scheduling benchmarks, all of which, chosen to test the performance of the algorithm, are NP-hard and cannot be solved to optimality within polynomially-bounded time. To apply the Bees Algorithm for machine scheduling, a new neighbourhood structure is defined. Several local search algorithms are combined with the Bees Algorithm.

This work also introduces an enhanced Bees Algorithm. Several additional features are considered to improve the efficiency of the algorithm such as negative selection, chemotaxis, elimination and dispersal which is similar to the 'site abandonment' strategy used in the original algorithm, and neighbourhood change. A different way to deploy neighbourhood procedures is also presented.

Three categories of machine scheduling problems, namely, single machine with a common due date, total weighted tardiness, and total weighted tardiness with sequence-dependent setup are used to test the enhanced Bees Algorithm's performance. The results obtained compare well with those produced by the basic version of the algorithm and by other well-known techniques.

ACKNOWLEDGEMENTS

I am privileged to have **Professor D.T.Pham** as my supervisor. The high standard of his research has always been an inspiration and a goal to me. I am deeply grateful to him for his consistent encouragement, invaluable guidance and strong support during the course of this study. His thoughtful advice and constant support extended to me will always be remembered.

I would like to express my sincere gratitude to my supervisor **Dr. Michael Packianather** for his huge support during my academic life in Cardiff. I am also very grateful to all the members of Manufacturing Engineering Centre, especially my **Cardiff Bay Bees Colleagues** for their friendship and support.

I would like to express my grateful appreciations and thanks to the **Royal Thai Government**, Thailand for supporting by granting a full PhD scholarship.

I also want to express my warmest thank to **Assoc. Prof. Dr. Kittichai Lavangnananda** and **his wife** for their support and encouragement. I also want to thank **Dr. Mario Javier Gonzalez Romo** and **Mr. Werachart Jantateimee** for their valuable help and support.

I wish to express my heartfelt gratitude to my mother, my late father and grandparents, and my cousins for all the love and support they have given to me.

DECLARATION

This work has not previously been accepted in substance for any degree or award at this or any other university or place of learning, nor is being submitted concurrently in candidature for any degree or other award.

Signed ..... (Janyarat Phrueksanant) Date ……

STATEMENT 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy (PhD).

Signed ..... (Janyarat Phrueksanant) 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 ..... (Janyarat Phrueksanant) 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 ..... (Janyarat Phrueksanant) Date ……

Content

List of Figures

List of Tables

Abbreviations

PSO	Particle Swarm Optimisation
DPSO	Discrete Particle Swarm Optimisation
ACO	Ant Colony Optimisation
DE	Differential Evolution
DDE	Discrete Differential Evolution
BA	Bees Algorithm
SA	Simulated Annealing
GAs	Genetic Algorithms
TS	Tabu Search
AIS	Artificial Immune System
CLONALG	Clonal Selection Algorithm
JIT	Just In Time
ES	Evolution Search
TA	Threshold Accepting
TAR	TA with a back step
SPV	Smallest Position Value
HTG	Tabu Search + Genetic Algorithms
HGT	Genetic Algorithms + Tabu Search
RDD	Relative range of due dates
TF	Tardiness Factor

BFOA	Bacterial Foraging Optimisation Algorithm
ATCS	Apparent Tardiness Cost with Setups
VNS	Variable Neighbour Search
RVNS	Reduced Variable Neighbourhood Search
ABA	The Adaptive Bees Algorithm
GVNS	General Variable Neighbourhood Search

List of Symbols

p	processing time
r	release date
d	due date
w	weight
p	Processing time
r	Release date
d	Due date
w	Weight
m	Machine
j	Job j
i	Job i
$C_{\max}$	Makespan
$L_{\max}$	Maximum Lateness
C_{j}	Completion time
$e^{-\Delta/c}$	Probability function
n	scout bees
m	number of selected sites
e	elite site
nsp	number of bees recruited for each non-elite site
nep	number of bees recruited for each elite site
ngh	neighbourhood size

E	Earliness
T	Tardiness
d	Due date
S	Solution / Sequence
α_i	Earliness penalty
β_i	Tardiness penalty
p_j/α_j	increasing ratios
p_j/β_j	non-decreasing ratios
F_{eBA}	The fitness function values of the enhanced BA
F_{ref}	The reference fitness function
R	Total number of runs
$\Delta_{\min}$	Minimum percentage of relative deviation
$\Delta_{\max}$	Maximum percentage of relative deviation
Δ_{std}	Standard deviation of percentage of relative deviation

CHAPTER 1

Introduction

1.1 Motivation

Combinatorial optimisation is optimisation in the case of discrete alternatives. Being positioned at the interface between mathematics, computer science, and operations research, the field of combinatorial optimisation has a diversity of algorithm approaches. Job scheduling, a combinatorial problem, is a process that is used on a regular basis in many companies. It deals with the allocation of resources to tasks over a given time period and its goal is to optimise some performance measure. Job scheduling plays an important role in most manufacturing and production systems as well as a number of information processing environments. It is also important in transportation and distribution settings.

In a manufacturing environment, the scheduling function has to interact with other decision making methods. Many computational methods such as Discrete Particle Swarm Optimisation (DPSO), Ant Colony Optimisation (ACO), and Discrete Differential Evolution (DDE) have been employed to solve job scheduling problems. More recently, the Bees Algorithm has become a possible new tool for job scheduling and other combinatorial optimisation problems.

The Bees Algorithm (Pham et al. 2005; Pham et al. 2006a, Pham et al. 2006b, Pham et al. 2006c, Pham et al. 2006d; Pham et al. 2007a) is an intelligent optimisation tool which is inspired by the natural foraging behaviour of honey bees. The algorithm employs a combination of global exploration and local exploitation. However, the Bees Algorithm was basically developed for solving continuous problems. In 2007, the use of the Bees Algorithm for a combinatorial problem was presented (Pham et al 2007). The algorithm successfully solved a machine scheduling with a common due date.

This work presents a hybrid algorithm. The Bees Algorithm is enhanced to increase its performance in solving different kinds of machine scheduling problems. All benchmarks used are known as NP-hard. The motivation for this research was to test how robust and efficient the Bees Algorithm was at handling such NP-hard problems.

1.2 Research Aims and Objectives

The overall aim of this research was to develop and improve swarm-based optimisation algorithms inspired by the foraging behaviour of honeybees and use the developed algorithms to solve various single machine scheduling problems.

The main research objectives were:

- To survey existing tools used to solve machine scheduling problems
- To study different types of machine scheduling problems and their characteristics
- To develop and enhance the Bees Algorithm with new features to overcome the drawbacks of its original version and enable it to solve machine scheduling problems
- To compare the results obtained with other optimisation methods

1.3 Research Methodology

To achieve the objectives, the following methodology was adopted:

- Literature review: The most relevant papers were reviewed to clarify the key points in the subject. Their advantages and disadvantages will be discussed in the thesis.
- A swarm-based optimisation procedure was proposed along with its enhanced version.

- The performance of the new versions of the algorithm was evaluated on a number of machine scheduling problems. In each case, performance measures were computed to assess the effectiveness of the new methods and comparisons with the original version and other optimisation methods were also carried out.

1.4 Outline of thesis

Chapter 2: In this chapter, definitions of machine scheduling problems and a review of the proposed engineering methodologies are given. Intelligence swarm-based optimisation algorithms including honeybee-inspired algorithms for combinatorial optimisation and neighbourhood search procedures are also reviewed.

Chapter 3: The Bees Algorithm to solve the problem of single-machine scheduling with common due date is introduced. This version is an enhancement of the basic version focusing on selecting the most promising solutions for the next generation. More neighbourhood procedures are deployed to increase search performance. The performances of the basic and improved algorithms are compared and the differences discussed. Also, the results from the improved algorithm are compared with those produced by well-known algorithms to show its performance and robustness.

Chapter 4: This chapter focuses on implementation of the Bees Algorithm to minimise the total weighted tardiness in single-machine scheduling. The disadvantages of the basic version are studied and an enhanced algorithm is proposed. The foraging behaviour of E. coli is used to help the main algorithm when it is trapped at local minima. The performances of the basic and improved algorithms are evaluated. Their results are also compared with those of other optimisation techniques.

Chapter 5: This chapter presents an application of the Bees Algorithm to solve the problem of scheduling for minimum total weighted tardiness with sequence-dependent setup times. The Apparent Tardiness Cost with Setups (ATCS) heuristic is applied to create a reasonably good starting solution. Neighbourhood change in Variable Neighbourhood Search (VNS) is adapted. The results obtained are compared with those of other existing optimisation techniques.

Chapter 6: This chapter presents the main contributions of this research and suggestions for future work in this field.

CHAPTER 2

BACKGROUND

2.1 Job Scheduling

Job scheduling problems involve solving for the optimal schedule under various objectives, different machine environments and characteristics of the jobs. In the definitions, job can be made up of any number of tasks. It can be considered as making a product. Basic information associated with a job are processing time (p_j), release date (r_j), due date (d_j), and weight (w_j). Processing time (p_j) represents the processing time of job j on a machine i. Release date (r_j) is the time that the job arrives at the system. It may also be referred to as the ready date. Due date (d_j) represents the committed shipping or completion date. Completion of a job after its due date is allowed, but then a penalty is incurred. Weight (w_j) represents the actual cost, which could be a holding or inventory cost.

The main possible machine environments are:

Single machine: only one machine is available to process jobs. Each job has single task. Every job is processed on the same machine.

Parallel machine: Multiple machines are available to process jobs. A job requires a single operation and can be processed on any machine.

Flow shop: There are a series of machines (m). Each job has exactly m tasks. The first task of every job has to be processed on machine 1, then on the machine 2 and so on. Every job goes through all m machines in a unidirectional order.

Job shop: There are m machines and j jobs. Each job has its own predetermined route to follow. A distinction is made between job shops in which each job visits each machine at most once and job shops in which a job may visit each machine more than once.

Examples of possible objective functions to be minimised are:

Makespan (C_{max}): The makespan is equivalent to the completion time of the last job. C_{max} is defined as:

$$C_{max} = \max (C_1, C_2, C_3, \ldots, C_n) \quad \text{(Eq. 2.1)}$$

The objective of this problem is to minimise C_{max} or to minimise the completion time of the last job to leave the system. This criterion is usually used to measure the level of utilisation of the machine.

Maximum Lateness (L_{max}): The maximum Lateness (L_{max}) measures the worst violation of the due date. It can be defined as:

$$L_{max} = \max (L_1, L_2, L_3, \ldots, L_n) \quad \text{(Eq. 2.2)}$$

Total Weighted Completion Time ($\sum w_j C_j$): C_j denotes the completion time of the j^{th} job in a batch of n jobs given. The sum of the completion times is often referred to as the flow time. It is defined as:

$$\sum_{j=1}^{n} C_j \quad \text{(Eq. 2.3)}$$

W_j denotes the weight assigned to j^{th} job in a batch of n jobs given. The total weighted completion time is defined as:

$$\sum_{j=1}^{n} w_j C_j \qquad \text{(Eq. 2.4)}$$

The total weighted completion time is then referred to as the weighted flow time. It gives an indication of the total holding or inventory costs incurred by the schedule. The objective of this problem is to minimise the total weighted completion time.

Total Weighted Tardiness ($\sum w_j T_j$): Total weighted tardiness is a more general cost function than the total weighted completion time. However, it is one of the strongly NP-hard problems which can be defined as:

$$\sum_{j=1}^{n} w_j T_j \qquad \text{(Eq. 2.5)}$$

All objective functions mentioned above are so-called regular performance measures which is a function that is non-decreasing in C_1, …, C_n. Recently objective function that are not regular has been studied. For example, when job j has a due date d_j, it may be subject to an earliness penalty, where the earliness of job j is defined as:

$$E_j = \max(d_j - C_j, 0) \quad \text{(Eq. 2.6)}$$

An objective such as the total earliness plus the total tardiness is defined as:

$$\sum_{j=1}^{n} E_j + \sum_{j=1}^{n} T_j \quad \text{(Eq. 2.7)}$$

A more general objective that is not regular is the total weighted earliness plus the total weighted tardiness:

$$\sum_{j=1}^{n} w'_j E_j + \sum_{j=1}^{n} w''_j T_j \quad \text{(Eq. 2.8)}$$

The weight associated with the earliness of job *j* may be different from the weight associated with the tardiness of job *j*. This problem is harder than the total tardiness problem (Lenstra 1977; Pinedo 2008; Robert and Vivien 2010).

2.2 Job Scheduling Solvers

This section presents existing techniques that have been successfully applied to job scheduling problems.

2.2.1 Simulated Annealing

Simulated Annealing (SA) was developed by Kirkpatrick et al. (1983) and Cerny (1985). The idea of SA algorithm was taken from the simulation of the annealing of solids. It has been successfully applied to many practical problems as it has a stochastic component, which facilitates a theoretical analysis of their asymptotic convergence. General schema for a SA algorithm to solve scheduling problem starts by generating a starting solution *S*. Then the neighbourhood of *S* is chosen randomly (*S*'). If the objective function value of *S*' is smaller than that of *S*, the new solution becomes the actual one and the search process is then continued from *S*'. On the other hand, if the objective function value of *S*' is greater than *S*, then *S*' is accepted as the actual solution with probability $e^{-\Delta/c}$, where c represents the actual value of the control parameter (temperature). At the beginning, the algorithm starts with a relatively high value of c so that most of the interior neighbourhood solutions are accepted. The c value is usually kept constant for a number of iterations and then reduced afterwards, so that the acceptance probability of inferior solution is relatively small in the end phase of search process. Fig 2.1 shows general pseudo code of a SA algorithm.

Step 1: Generate a starting solution S as initial solution $S_{best} = S$

Step 2: Determine a starting temperature c

Step 3: While

Choose a random neighbour S' of current solution

Set $\Delta = f(S') - f(S)$

If $\Delta \leq 0$ then $S = S'$

If $f(S) < f(S_{best})$ then $S_{best} = S$

Else if $e^{-\Delta/c} >$ random [0,1] then S = S'

End

Lower the temperature c

End

Step 4: If stopping criterion not met then goto step 3

Figure 2.1 General pseudo code of a Simulated Annealing algorithm

2.2.2 Genetic Algorithms

Genetic Algorithms (GAs) are invented by Holland (1975). The algorithms have been used for a wild variety of problems including machine learning, game playing, and combinatorial optimisation. GAs use a population of possible solutions to conduct a robust search of search space. Initially, a set of solutions is generated randomly. Each of which is then evaluated by fitness function. The algorithm then enters a loop. Any iteration in the loop is called a generation, which consists of two steps: selection and recombination. Holland (1975) suggested that the solutions with better fitness values should have a higher probability to be selected for reproduction. In recombination step, the most common operators are crossover and mutation. Results from recombination operators are the population for the next generation. The loop continues until a stopping criteria is met (De Jong 2006; Goldberg 1989; Webster et al 1998). Figure 2.2 shows general pseudo code of a GAs.

Step 1: Create an initial population of *m* parents

Step 2: Compute and save the fitness value *f*(*i*) for each individual (*i*)

Step 3: Define selection probabilities *p*(*i*) for each parent *i*

So that *p*(*i*) is proportional to *f*(*i*)

Step 4: Generate *m* offspring by probabilistically selecting parents to produce offspring

Step 5: Select only the offspring to survive

Step 6: Repeat step 2 until a stopping criterion has been met

Figure 2.2 General pseudo code of a Genetic algorithms

2.2.3 Tabu Search Algorithm

Tabu search (TS) is a meta-heuristic that guides a local search procedure to explore the solution space beyond local optimality (Glover and Laguna 1997). In order to improve the efficiency of the exploration process, local information and some information related to the exploration process must be memorised. This adaptive memory usage is an essential feature of TS.

The TS begins by marching to a local minima. To avoid retracting the steps used, the method records recent moves in one or more tabu lists. The original intent of the list was not to prevent a previous move from being repeated, but rather to insure it was not reversed. The tabu lists are historical in nature and form the tabu search memory. The role of the memory can change as the algorithm proceeds. At initialisation the goal is to make a coarse examination of the solution space, known as diversification, but as candidate locations are identified the search is more focused to produce local optimal solutions in a process of intensification. In many cases the differences between the various implementations of the tabu method have to do with the size, variability, and adaptability of the tabu memory to a particular problem domain.

The TS has traditionally been used on combinatorial optimisation problems. The technique is straightforwardly applied to continuous functions by choosing a discrete encoding of the problem. Many of the applications in the literature

involve integer programming problems, scheduling, routing, traveling salesman and related problems.

2.2.4 Ant Colony Optimisation

Ant Colony Optimisation (ACO) was introduced by Dorigo et al. (1991). The ACO is a non-greedy population-based meta-heuristic which emulates the behaviour of real ants. Ants are capable of finding the shortest path from the food source to their nest using a chemical substance called pheromone, which is used to guide the exploration. The pheromone is deposited on the ground as the ants move and the probability that a passing stray ant will follow this trail depends on the quantity of pheromone laid (Bilchev and Parmee 1995).

Current applications of ACO algorithms fall into the two important problem classes of static and dynamic combinatorial optimisation problems. The artificial ants in ACO implement a randomised construction heuristic which makes probabilistic decisions as a function of artificial pheromone trails and possibly available heuristic information based on the input data of the problem to be solved. As such, ACO can be interpreted as an extension of traditional construction heuristics, which are readily available for many combinatorial optimisation problems (Dorigo et al. 1999; Dorigo 2004; Bonabeau et al. 1999; Pan et al. 2010). Figure 2.3 shows the pseudo code of ACO.

Step 1: Initialise pheromone values

Step 2: While (stopping criterion not met) do

Step 3: Create all ants solutions

Step 4: Perform local search

Step 5: Update pheromone values

Step 6: End while

Figure 2.3 The pseudo code of Ant Colony Optimisation

2.2.5 Discrete Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) is a population based meta-heuristic proposed by Kennedy and Eberhart (1995). It is based on the social behaviour of groups of organisations, for example the flocking of birds or the schooling of fish and originally designed for continuous optimisation domains. PSO deploys the exploring agents called particles that can adjust their positions in time according to their own experience and to other particles' experience (Eberhart and Kennedy 2001).

Discrete Particle Swarm Optimisation (DPSO) was first proposed by Kennedy and Eberhart (1997). DPSO approach differs both for the way it associates a particle position with a discrete solution and for the velocity model used. Several studies have applied the DPSO approach to combinatorial optimisation problem such as the travelling salesman problem, vehicle routing problem, and job scheduling problems. The pseudo code of DPSO is given in Fig.2.4.

Step 1: Create particles (population)

Step 2: While (stopping criterion not met) do

Step 3: Evaluate each particle's position according to the objective function

Step 4: Find the personal best

Step 5: Update the personal best

Step 6: Find the global best

Step 7: Update the global best

Step 8: Update particles' velocities

Step 9: Move particles to their new position according to their velocity

Step 10: Go to step 3 until stopping criterion has been met

Figure 2.4 The pseudo code of Discrete Particle Swarm Optimisation

2.2.6 Discrete Differential Evolution

Differential Evolution (DE) was introduced by Storn and Price (1997). DE is a stochastic population-based heuristic that has been applied on many numerical optimisation problems. The standard DE algorithm is given in Figure 2.5. Recently, Discrete Differential Evolution Algorithm (DDE) was proposed to solve complex combinatorial optimisation problems with discrete decision variables such as the traveling salesman and job scheduling problems. The advantages of DDE include a simple structure, immediately accessible for practical applications, ease of implementation, speed to acquire solutions, and robustness. However, the application of DDE on combinatorial optimisation problems are still considered limited.

Step 1: Initialise parameters and population

Step 2: Evaluate population

Step 3: Do

Step 4: Obtain mutant population

Step 5: Obtain trial population

Step 6: Evaluate trial population

Step 7: Make selection

Step 8: Apply local search (optional)

Step 9: While (not termination)

Figure 2.5 The standard pseudo code of Differential Evolution Algorithm

2.2.7 Exact Algorithm

In 2008, Tanaka and Fujikuma (2008) have proposed an Exact Algorithm solve general single machine scheduling without machine idle time problem. It is based on Successive Sublimation Dynamic Programming (SSDP) method. Its process starts from a relaxation of the original problem. Thus Langrangian Relaxation (LR) technique is employed. Three relaxations (LR_1), $(\widehat{LR_2})$ and $(\widehat{LR_2^m})$ are generated. The algorithm composes of three stages: (LR_1) is solved first, $(\widehat{LR_2})$ is solved next and then $(\widehat{LR_2^m})$ is solved. The constrains are successively added for better relaxations during the main loop of the SSDP method until the gap between lower and upper bounds becomes zero. Reduction of memory usage is also performed by network reduction techniques (Tanaka et al 2009; Tanaka and Fujikuma 2012). Recently, an Exact Algorithm has been successfully applied to solve several types of single machine scheduling problems such as the single-machine earliness-tardiness scheduling problem (Tanaka 2012), the precedence-constrained single-machine scheduling problem (Tanaka and Sato 2013), AND the single-machine total weighted tardiness problem with sequence-dependent setup times (Tanaka and Araki 2013).

2.3 Artificial Immune System

The natural immune system is a very complex system with several mechanisms to defence against pathogenic organisms. However, the natural immune system is also a source of inspiration for solving optimisation problems. From the information processing perspective, immune system is a remarkable adaptive system and can provide several important aspects in the field of computation. When incorporated with evolutionary algorithms, immune system can improve the search ability during the evolutionary process. The Artificial Immune Systems (AIS) are machine-learning algorithms that embody some of the principles and attempt to take advantages of the benefits of natural immune systems to deal with complex problem domains. Some of theories primarily used in AIS are briefly described below:

The Clonal Selection Principle describes the basic characteristics of an adaptive immune response to an antigenic stimulus. Only those cells that able to recognise an antigenic stimulus will proliferate and differentiate into effector cells and will be selected. The main features of clonal selection theory are cloning, elimination and proliferation (de Castro and Timmis 2002; Aickelin and Dasguta 2005). The Clonal Selection Algorithm (CLONALG), the most well-known AIS algorithm, proposed by de Castro and Von Zuben (2002), is one such system inspired by the clonal selection theory of acquired immunity, which has shown success on broad range of engineering problem domains.

The Immune Network Theory was proposed by Jerne (1974). The immune network was introduced as a fundamental idea to explain phenomena like repertoire selection, tolerance, self/nonself discrimination and memory (Varela and Coutinho 1991). The hypothesis was that antibody molecule could be recognised by a set of other antibody molecules. A regulated network of molecules and cells that recognise one another even in the absence of antigens composes the immune system.

The Negative Selection describes the process whereby a lymphocyte-antigen interaction results in the death of that lymphocyte (de Castro and Von Zuben 2002). During the generation of T-cells, T-cells that react against self-proteins are destroyed. Only T-cells that do not bind to self-proteins are allowed to leave the thymus then circulate throughout the body to protect the body from foreign antigen (Aickelin and Dasguta 2005).

The Positive selection serves the purpose of avoiding the accumulation of useless lymphocytes. In positive selection of T-cells, all T-cells must recognise antigens associated with self-MHC molecules. Only those of T-cells that capable of binding to Self-MHC (Major Histocompatibility Complex) molecules can survive. The positive selection algorithm consists of three main processes: generation of the potential repertoire of immature T-cells, Affinity evaluation and generation of available repertoire (de Castro and Timmis 2002; Zhang and QI 2012).

The Danger Theory was proposed by Matzinger (1994). The key why the immune system is able to distinguish between the nonself-antigens and the self-antigens is that the nonself-antigens make the body produce biochemical reactions different from natural rules and the reactions will make the body produce danger signals of different levels. Thus, the immune system produces danger signals based on the environmental changes and then leads to the immune responses. In essence, the danger signal creates a danger zone around itself and immune cells within this danger zone will be activated to participate in the immune response. The Danger theory explains the immune response of the human body by the interaction between antigen presenting cells and various signals (Zhange et al. 2013; Lu 2012; Aickelin and Dasguta 2005; Matzinger 2002)

The Artificial Immune system was introduced as a new computational intelligent paradigm. It is a general framework for a distributed adaptive system and could be applied to many problem domains such as Network Intrusion Detection problem (Kim and Bentley 1999), Autonomous Navigation (Watanabe et al 1999), Computer Network Security (Hofmeyr and Forrest 2000), Job Scheduling (Coello et al. 2003; Hart and Ross 1999; Lee and Zomaya 2007), Data Analysis and Optimisation (de Castro and Von Zuben 2001; de Castro and Timmis 2002; Zhang and QI 2012). It represents a powerful technique that already emerged.

2.4 The Honeybees-inspired Algorithm

2.4.1 The Honeybees in nature

A colony of honeybees can extend itself over long distances and in multiple directions simultaneously to exploit a large number of food sources (Von Frisch 1967; Seeley 1996). A colony prospers by deploying its foragers to good fields. In principle, flower patches with plentiful amounts of nectar or pollen that can be collected with less effort should be visited by more bees, whereas patches with less nectar or pollen should receive fewer bees (Camazine et al. 2003).

The foraging process begins in a colony by scout bees being sent to search for promising flower patches. Scout bees move randomly from one patch to another. During the harvesting season, a colony continues its exploration, keeping a percentage of the population as scout bees (Seeley 1996).

When they return to the hive, those scout bees that found a patch which is rated above a certain quality threshold (measured as a combination of some constituents, such as sugar content) deposit their nectar or pollen and go to the "dance floor" to perform a dance known as the "waggle dance" (Von Frisch 1967). Source quality can be understood as simply the relation between gain and cost from a specific nectar source (Von Frisch 1967).

This mysterious dance is essential for colony communication, and contains three pieces of information regarding a flower patch: the direction in which it will be found, its distance from the hive and its quality rating (or fitness) (Von Frisch 1967; Camazine et al. 2003). This information helps the colony to send its bees to flower patches precisely, without using guides or maps. Each individual's knowledge of the outside environment is gleaned solely from the waggle dance. This dance enables the colony to evaluate the relative merit of different patches according to both the quality of the food they provide and the amount of energy needed to harvest it (Camazine et al. 2003). After waggle dancing on the dance floor, the dancer (i.e. the scout bee) goes back to the flower patch with follower bees that were waiting inside the hive. More follower bees are sent to more promising patches. This allows the colony to gather food quickly and efficiently.

While harvesting from a patch, the bees monitor its food level. This is necessary to decide upon the next waggle dance when they return to the hive (Camazine et al. 2003). If the patch is still good enough as a food source, then it will be advertised in the waggle dance and more bees will be recruited to that source.

Nectar source selection behaviour is one of the most challenging as well as vital tasks for honey-bee colonies (Camazine et al. 2003). When a honey-bee colony becomes overcrowded it needs to be divided for effective source management (Von Frisch 1967; Camazine et al. 2003). This critical decision making process works without a central control mechanism. Nectar source selection behaviour mainly deals with the situation of a colony choosing between several nectar

sources by simply measuring several factors at once and comparing them with other solutions. The decision is made when all the scout bees are dancing for the same site and it takes a couple of days before half of the colony moves to a new hive (Camazine and Sneyd 1991; Camazine et al. 1999: Seeley and Visscher 2003).

2.4.2 Artificial Bee Colony Algorithm

Artificial Bee Colony (ABC) is a swarm-based algorithm that was originally proposed by Karaboga (2005); Karaboga and Basturk (2007). It simulates the foraging behaviour of a honeybee swarm. In its basic version, honeybees are classified into three groups namely, employed bees, onlookers, and scouts. An employed bee is responsible for searching for food source and collecting nectar. An onlooker waits in the hive and decides on whether a food source is acceptable or not after watching employed bees perform waggle dance. A scout searches for new food source randomly. The main steps of the ABC algorithm are given in Figure 2.6 (Karaboga (2005); Karaboga and Basturk (2007); Karaboga and Basturk (2008).

Step 1: Send the scouts onto the initial food sources

REPEAT

Step 2: Send the employed bees onto the food sources and determine their nectar amounts. Calculate the probability value of the sources with which they are preferred by the onlooker bees .

Step 3: Stop the exploitation process of the sources abandoned by the bees

Step 4: Send the scouts into the search area for discovering new food sources, randomly

Step 5: Memorise the best food source found so far UNTIL (requirements are met)

UNTIL (requirements are met)

Figure 2.6 The main steps of the ABC algorithm

Later a Discrete Artificial Bee Colony (DABC) algorithm was proposed to solve job scheduling problems for examples, the lot-streaming flow shop scheduling problem, the multi-objective flexible job-shop scheduling problem with maintenance activities, and the flexible job-shop scheduling problem (Pan et al. 2010; Li et al. 2013; Thammano and Phu-ang 2013). The DABC algorithm represents a food source as a discrete job permutation and applies discrete operators to generate new neighbouring food sources for the employed bees, onlookers and scouts.

2.4.3 The Bees Algorithm

2.4.3.1 The Bees Algorithm for continuous domains

The Bees Algorithm (BA) was developed by a group of researchers at the Manufacturing Engineering Centre, Cardiff University (Pham et al. 2005; Pham et al. 2006a, Pham et al. 2006b, Pham et al. 2006c, Pham et al. 2006d; Pham and Ghanbarzadeh 2007; Pham et al. 2007a). This algorithm emulated the behaviour of honeybees in foraging for pollen and nectar. The algorithm required parameters, namely the number of scout bees (*n*), number of selected sites (*m*), number of top-ranking (*elite*) sites among the *m* selected sites (*e*), number of bees recruited for each non-elite site (*nsp*), number of bees recruited for each elite site (*nep*), and neighbourhood size (*ngh*). The optimisation process started with *n* scout bees randomly spread across the solution space. Each scout bee was associated with a possible solution to the problem. The solutions were evaluated and ranked in descending order of the fitness, and the best *m* sites were selected for neighbourhood search.

In the neighbourhood search procedure, more forager bees were sent in the neighbourhood of the elite (*e*) sites, and fewer bees around the non-elite (*m-e*) sites. According to this strategy, the foraging effort was concentrated on the very best (i.e., elite) solutions. That is, *nep* bees were sent to forage around the elite sites, while the area around the non-elite locations was exploited by *nsp* bees. Within the given neighbourhood area (i.e., flower patch size), some of the newly generated solutions were expected to be better than that found by the scout bees.

In the global search procedure, the unselected scout bees (*n*-*m*) were used to explore at random the solution space. This kind of search was to avoid bees being trapped at local optima. At the end of each cycle, a new list of scout bees was formed, comprising the fittest solutions from each neighbourhood (neighbourhood search results), and the new randomly generated solutions (global search results). This list would be sorted in the next iteration and used for a new phase of optimisation. The combination of exploitative (neighbourhood) and explorative (global) search would be able to capture the best solution quickly and efficiently. These steps were repeated until the stopping criterion was met (Ghanbarzadeh 2007). The pseudocode of the BA and the algorithm flowchart for continuous domains is shown in Figures 2.6 and 2.7 respectively (Pham et al. 2006b; Ahmad 2012).

Step 1: Initialise population with random solutions

Step 2: Evaluate fitness of the population

Step 3: While (stopping criterion not met)

//Forming new population

Step 4: Select sites for neighbourhood search

Step 5: Recruit bees for selected sites (more bees for best e sites)

and evaluate the fitness

Step 6: Select the fittest bee from each patch

Step 7: Assign remaining bees to search randomly and evaluate their fitness

Step 8: End While

Figure 2.7 The pseudocode of the Bees Algorithm for continuous domains

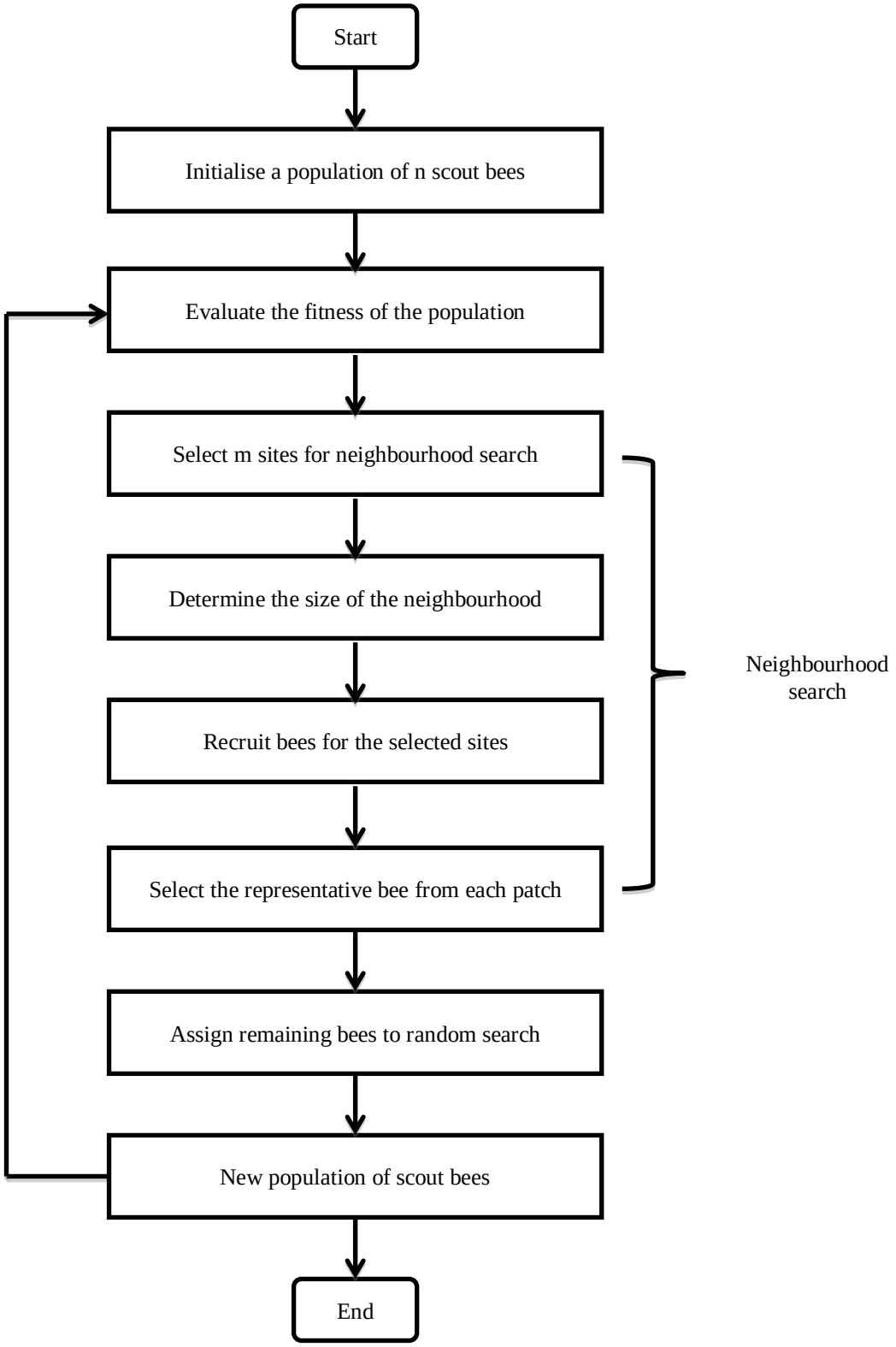


Figure 2.8 Flowchart of the Bees Algorithm

2.4.3.2 The Bees Algorithm for Job Scheduling Problems

In 2007, the first Bees Algorithm to solve machine scheduling was proposed (Pham et al. 2007b). This work is the first to report the application of the Bees Algorithm to a combinatorial problem. The pseudo-code of the Bees Algorithm for scheduling problem is given in Fig. 2.8. In essence, the algorithm is very similar to the original algorithm. The main differences here are: in step 5, the patch idea is replaced by a local search operator to be able to perform a local search and the, shrinking procedure is also removed from the algorithm. However, the abandonment procedure is kept to help the algorithm to improve the global search part.

The main feature of combinatorial domains, unlike continuous domains, is that there is no mathematical distance definition for the neighbourhood search. Since the Bees Algorithm was developed for continuous domains, it is necessary to modify the neighbourhood part by simply replacing the patch with a local search operator (Koc 2010).

There are several exchange neighbourhood strategies for examples, 2-Opt, 3-Opt, and Swap operators. 2-Opt was first proposed by Croes 1958 for solving the traveling salesman problem. The main idea is to break two edges and reconnect them in other way. There is also 3-Opt approach, a cut of 3 points and reconnect them in other possible ways. The same problem may have multiple different neighbourhoods defined on it, local neighbourhood search that involve changing

Step 1: Initial population with n random solution; random(Sequence(n)).

Step 2: Evaluate fitness of the population.

Step 3: While (stopping criterion not met)

Step 4: Select sites (*m*) for neighbourhood search.

Step 5: Recruit bees for selected sites (more bees for best e sites), evaluate fitnesses, select the fittest bee from each site and shrink patches

for (*k*=1 ; *k*=e ; *k*++) // Elite Sites

for (*i*=1 ; *i*= nep ; *i*++) // More Bees for Elite Sites

RecruitedBee(*k*)(*i*) = NeighbourhoodOperator(Sequence(*k*));

Evaluate Fitness = RecruitedBee(*k*)(*i*);

//Evalute the fitnees of recruited Bee(*i*)

If (Bee(*i*) is better than Bee(*i*-1)) RepresentativeBee = RecruitedBee(*k*)(*i*);

for (*k*=*e* ; *k*=*m* ; *k*++) // Other selected sites (*m*-*e*)

for (Bee=1 ; Bee= *nsp* ; Bee++) // Less Bees for Other Selected Sites (*m*-*e*)

RecruitedBee(*k*)(*i*) = NghOperator(Sequence(*k*));

Evaluate Fitness = RecruitedBee(*k*)(*i*);

//Evalute the fitnees of recruited Bee(*i*)

If (Bee(*i*) is better than Bee(*i*-1)) RepresentativeBee = RecruitedBee(*k*)(*i*);

Step 6: If (Iteration > sat)

If (no improvement on the site)

Save the Best Fitness; Abandon the Site;

Bee(*m*) = GenerateRandomValue(All Search Space);

Step 7: Assign remaining bees to search randomly and evaluate their fitnesses. // (*n*-*m*) assigned to search randomly into whole solution space

Step 8: End while

Figure 2.9 The pseudo-code of the BA for scheduling problem (Koc 2010)

up to *k* components of the solution is often referred to as k-opt. Swap and insert operators are considered as neighbourhood strategies. They simply change the position of a randomly selected node to create an altered path. In swap neighbourhood, two nodes are interchanged whereas in insert neighbourhood, one node is removed from its current position and then inserted elsewhere (Aarts and Lenstra 1997). In Pham et al. 2007b, only the exchange, 2-Opt and 3-Opt were used to modify the Bees Algorithm.

2.5 Summary

This chapter briefly describes job scheduling problems and some existing techniques applied to solve those problems. It also provides general background of the Bees Algorithm for combinatorial domains as well as continuous domain.

CHAPTER 3

THE ENHANCED BEES ALGORITHMS WITH NEGATIVE SELECTION FOR SINGLE MACHINE WITH A COMMON DUE DATE

3.1 Preliminaries

Scheduling multiple jobs on a machine with a common due date set costs depend on whether a job is finished before or after the specified due date. Minimising earliness penalty such as inventory cost and tardiness penalty imposed by customers pushes the completion time of each job as close as possible to the due date. If the optimal sequence cannot be constructed without considering the value of the due date, the common due date is called restrictive. This problem is known to be intractable (Garey and Johnson 1979).

Two newly developed Bees Algorithm with Negative Selection based Artificial Immune System (AIS) are presented in this chapter. These algorithms are enhanced version of their basic counterpart for combinatorial problems to solve single-machine with common due date problem. The discrete uniform distribution technique is also used for randomly generating the idle time during initialisation when needed.

The chapter is organised as follows: Section 3.2 describes the single machine scheduling problem with a common due date, its model, its well-known properties and benchmark. Section 3.3 presents the enhanced Bees Algorithms for single machine with common due date. Their characteristics are also described. Results are tabulated in Section 3.4 and the summary of this work is in Section 3.5

3.2 Earliness and Tardiness penalties in single-machine problem with a common due date

Common due date problems have been studied extensively in recent years. Kanet (1981) is one of the pioneers studying common due date problems. This contribution has been extended in many directions; see, for examples, Baker and Scudder (1989a&b), Biskup and Cheng (1999), Hoogeveen and van de Velde (1991), Feldmann M, and Biskup D (2003), Hino et al. (2005), Pan et al. (2006), Nearchou (2006), Nearchou (2008), Pham et al. (2007b), and Talebi et al (2009).

This problem became important with the advent of the just in time (JIT) concept which is a production strategy that strives to improve the business return on investment by reducing costs. In the JIT scheduling environment, the product should be finished as close to due date as possible. An early job completion results in inventory carrying costs, such as storage and insurance costs. On the other hand, a tardy job completion results in penalties, such as loss of customer goodwill and damaged reputation. When scheduling on a single machine against common due date, one job at most can be completed exactly at the due date. Hence, some of the jobs have to be completed early while other jobs must be finished late.

3.2.1 The Earliness and Tardiness Model

The concept of earliness and tardiness (E/T) has spawned a rapidly developing line of research in scheduling area. Because the use of both earliness and tardiness penalties gives rise to non-regular performance measure, it has led to new methodological issues in the design of solution procedures. In the E/T problem, the set of jobs to be scheduled is known in advance and is simultaneously available. The vast majority of articles on E/T problems also deal with single machine models.

To describe an E/T model, let n be the number of jobs to be scheduled. Job i is described by a processing time p_i and a due date d_i. All jobs are assumed to be available at time Zero. If the completion time C_i of job i is smaller than or equal to

common due date d, which is assumed as given, the jobs' earliness is $E_i = (d_i. - C_i)$. Accordingly, a job i is tardy with tardiness $T_i = (C_i - d_i)$, if its completion time is greater than the common due date d. As it is not known in advance whether a job will be completed before or after due date, earliness and tardiness are calculated as is $E_i = \max\{0, d_i. - C_i\}$ and $T_i = \max\{0, C_i - d_i\}$ for all jobs $i = 1, \ldots, n$. The per time unit penalties of the job i for being early or tardy are α_i and β_i, respectively.

The basic E/T objective function for a schedule S can be written as *f(S)*, where

$$f(S) = \sum_{i=1}^{n} \alpha_i E_i + \sum_{i=1}^{n} \beta_i T_i \qquad \text{(Eq. 3.1)}$$

Some of E/T problems have been derived for models in which all jobs have a common due date ($d_i = d$) (Baker and Scudder 1989a; Baker and Scudder 1989b).

3.2.2 The Restrictive Common Due Date

The restrictive and unrestrictive cases are two main approaches to address the common due date. In the unrestricted case, the optimal schedule (S) can be constructed without considering the due date, which means it has no influence on the optimal sequence. However, if the due date is known and it affects the optimal sequence of jobs, then it is considered restrictive.

The restrictive common due date is NP-hard which has been proven independently by Hall et al. (1991) and Hoogeveen and Van de Velde (1991). Three well-known properties that are essential for an optimal schedule in the restrictive case are as follows:

1. There are no idle times between consecutive jobs (Cheng and Kahlbacher 1991).
2. An optimal schedule has the so-called V-shape property, that is, jobs finished before the due date are ordered according to non increasing ratios p_j/α_j and jobs finished after the due date are ordered according to non-decreasing ratios p_j/β_j (Smith 1956).
3. There is an optimal schedule in which either the processing time of the first job starts at time zero or one job is finished at the due date (Hoogeveen and Van de Velde 1991)

All potential optimal schedules can be divided into three cases:

1) The first job starts at time zero and the last early job is finished exactly at time *d*.

2) The first job starts at time zero and the last early job is finished before *d*, here a straddling job exists.

3) The first job does not necessarily start at time zero.

3.2.3 Benchmark for single machine with common due date problems

Biskup and Feldmann (2001) have developed a set of the restricted single machine with common due date benchmark. There are seven categories of problems with 10, 20, 50, 100, 200, 500, and 1000 jobs. Each category contains 10 instances. For each of the jobs, the individual processing times p_i, earliness α_i and tardiness β_i penalty are given. Four values of parameter *h*: 0.2, 0.4, 0.6, 0.8, are used to calculate more or less restrictive common due dates. Therefore this benchmark has 280 test instances in total. The common due date *d* is calculated by

$$d = round[\sum p_i * h] \qquad \text{(Eq.3.2)}$$

where round[*x*] gives the biggest integer, which is smaller than or equal to *x*

$\sum p_i$ denotes the sum of the processing times of the *n* jobs

These instances are available at OR-LIBRARY website: http://people.brunel. ac.uk/~mastjjb/ jeb/orlib/schinfo.html

3.3 The Enhanced Bees Algorithms for Single Machine with Common Due Date

In 2007, Pham et al. (2007b) has presented the Bees Algorithm to solve single machine with common due date. This work is the first to report the application of the Bees Algorithm to a combinatorial problem. In this basic version, two neighbourhood search methods, namely simple-swap and insert method are applied. The search of best idle time is considered as continuous domain. The computational results show that the Bees Algorithm performed more strongly than the existing techniques during that period of time.

The Bees Algorithm with Negative Selection proposed in this chapter is an enhanced version which aims to improve the basic Bees Algorithm in choosing the fittest solutions from selected patch sites after neighbourhood search. The basic version was studied and observed that keeping the fittest solution from each patch site might not always be a good option for single machine scheduling problem. There is a possibility that the algorithm will keep many of the same solutions which means each selected patch site sometimes might unintentionally produce the same sequences as other patch sites during neighbourhood search.

Moreover, there is a chance that the second best solution and sometimes as well as the third best solution from a patch site might have better fitness values than other sites' fittest one. Keeping duplicitous solutions for the next generation could cause high computational time as well as being struggled in local optima.

3.3.1 The enhanced Bees Algorithms' characteristics

In this section, three key features namely the Discrete Uniform Distribution, Neighbourhood Search Procedures, and Negative Selection based Artificial Immune System deployed to improve the Bees Algorithms' performance are presented.

3.3.1.1 The Discrete Uniform Distribution

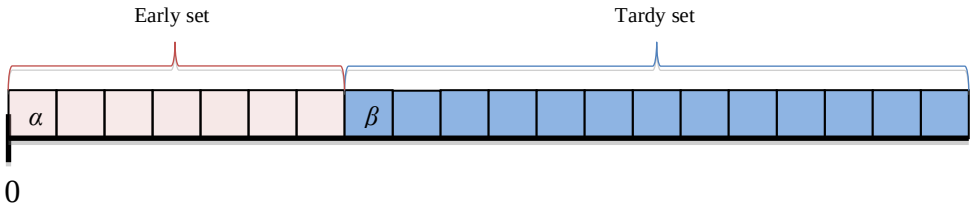
The discrete uniform distribution is the distribution in which all possible values have equal probabilities. The uniform distribution is characterised as follows:

A discrete random variable *R*, taking value 1,2,3,…,*n* such that

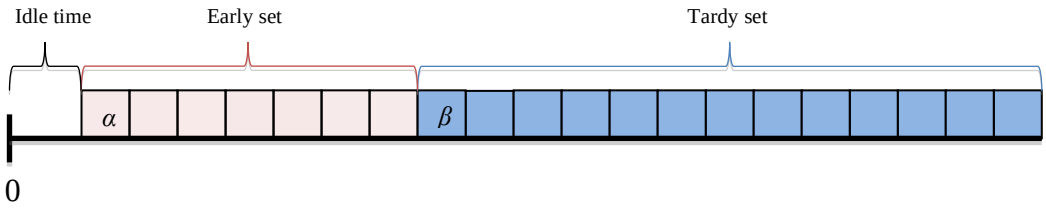
$$P(R = r) = \begin{cases} \frac{1}{k} & r = 1,2,3,\dots,k \\ 0 & otherwise \end{cases} \quad \text{(Eq.3.3)}$$

A random variable *R* used in this way, associated with the results rather than equal to them, can be a very useful concept. It is called a dummy variable or an indicator variable (Clarke and Cooke 2004).

According to Property 3, the search for an optimal schedule should not be restricted to sequences starting at time Zero. The discrete uniform distribution is used to randomly generate the idle time, which will be inserted at the beginning of the schedule only. Fig. 3.1 illustrates possible solution sets. Fig. 3.1 (a) shows a possible solution when first job starts at time zero. Fig. 3.2 (b) shows a possible solution with idle time when processing of first job is delayed.



(a) A possible solution without idle time inserted



(b) A possible solution with idle time inserted

Figure 3.1 Illustration of possible solution sets

3.3.1.2 Neighbourhood Search Procedures

Local search which is a widely used, is a general approach to solving hard optimisation problems. An optimisation problem has a set of solutions and an objective function that assigns a numerical value to every solution. Typically, local search procedures for job scheduling move from feasible schedules to feasible schedules. A key issue in these procedures is thus to design, or to select, moves that preserve feasibility in hope of improving an objective function which measures the quality of solutions to the problem at hand.

A very simple neighbourhood search is the Swap, a well-known local search method for combinatorial problems (Aarts and Lenstra 1997). In this enhanced Bees Algorithm's neighbourhood search step, two different types of swap methods are deployed. The first procedure is double swap method. Two jobs will be selected randomly regardless of whether these jobs are in early or tardy set and then swapped. The same process will repeat once again with two other jobs. Fig 3.2 shows double swap method deployed in neighbourhood search step of the enhanced Bees Algorithm when the first job starts at time zero. In the second neighbourhood search procedure shown in Fig. 3.3, two groups of jobs are selected randomly and then their positions are swapped. Also, early and tardy sets are not considered.

Another search method deployed in this enhanced Bees Algorithm is insert method. It is similar to simple-swap but insertion does not work vice versa. A

randomly selected job is simply inserted in a randomly defined position. It is slightly modified for this problem. Inserting can only occur between early and tardy sets. Fig 3.4 shows the third procedure deployed. A job from early set is randomly selected and then inserted into a position in tardy set. Fig 3.5 shows the fourth procedure. A job from tardy set is randomly selected and then inserted into a position in another set.

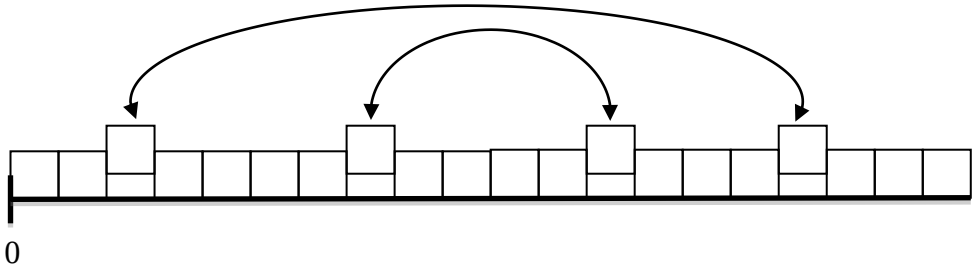


Fig 3.2 Double-swap method

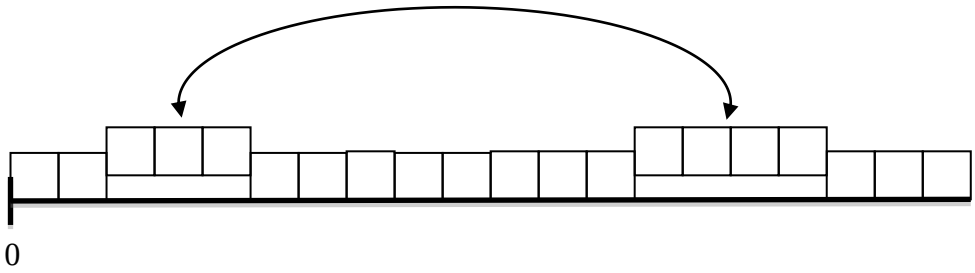


Fig 3.3 Two groups-swap method

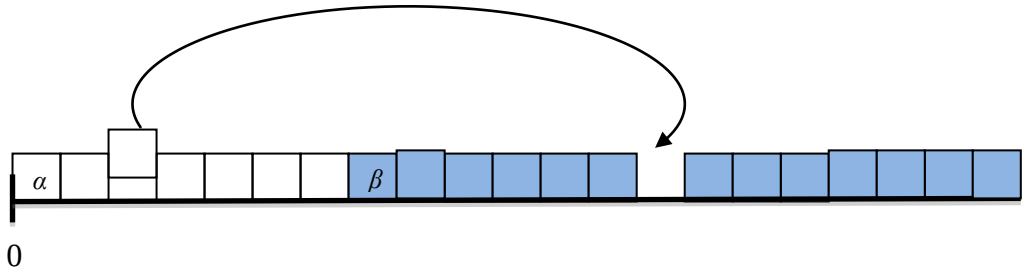


Fig 3.4 Insert method from early set to tardy set

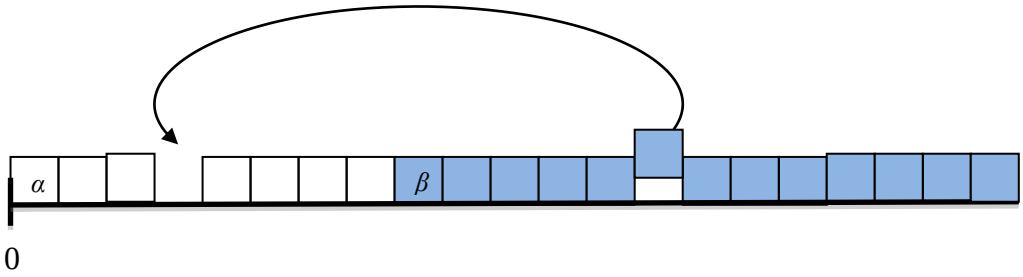


Fig 3.5 Insert method from tardy set to early set

3.3.1.3 Negative Selection

In recent years, attention has been drawn to Artificial Immune System (AIS), a biologically inspired computing paradigm. AIS abstracts and models tackle challenging problem in dynamic environments. Major AIS model include Positive Selection, Negative Selection, Clonal Selection, Danger Theory, and Immune Networks. This soft computing paradigm has been showing potential in job scheduling as well as other applications (Hart and Ross 1999; Coello et al. 2003; Aickelin et al. 2004; Chandrasekaran et al. 2006; Chen et al 2012).

A well known artificial Negative Selection scheme was proposed in Forrest et al. (1994). Three principles of the algorithm presented were defining self, generating detectors and monitoring the occurrence of anomalies. Fig 3.6 shows the negative selection algorithm proposed by Forrest et al. (1994). Strings are randomly generated and placed in a set P of immature T-cells. Then the affinity of all T-cells in P is determined with all elements of the self-peptides, named self-set S. If the affinity of an immature T-cell with at least one self-peptide is greater than or equal to a given cross-reactive threshold, then the T-cell recognises this self-peptide and has to be eliminated (negative selection), else the T-cell is introduced into the available repertoire A. The result showed that negative selection algorithm has been successfully applied to detect changes in computer systems that lead to improvement of system robustness (de Castro and Von Zuben 2002; de Castro and Timmis 2002).

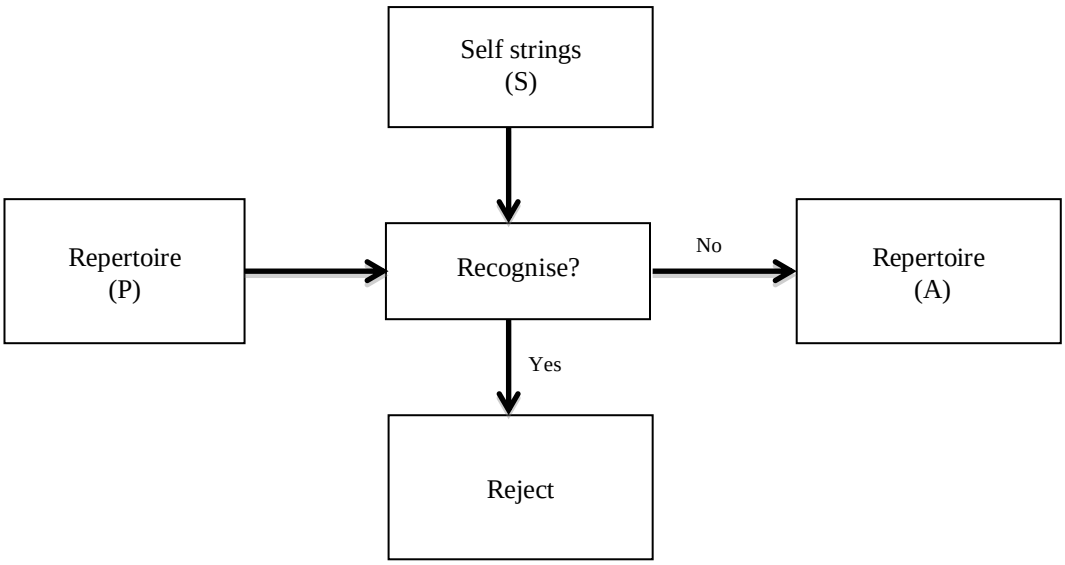


Figure 3.6 The negative selection algorithm

In the basic Bees Algorithms, after neighbourhood search, the fittest bee from each patch site will be saved for next iteration. In single machine scheduling, applying this idea often causes keeping duplicitous solutions and this does not guarantee that all best solutions are chosen as part of the population in the next generation. Also there is possibility that the second best solution from a selected site could have better fitness than the best solution from another site. To overcome this drawback, negative selection model is adapted for the Bees Algorithm. Two major phases of negative selection algorithm, detector generation and anomaly monitoring, are thus considered.

After neighbourhood search process, all solutions derived will be sorted and transferred into repertoire (P). First solution (a sequence) in repertoire (P) will be copied into Self-strings (S) and repertoire (A). Then next solution in repertoire P will be considered by matching it with strings in Self-strings. If it is recognised then it will be eliminated. If not, it will be introduced into repertoire A. In the opposite sense, if the set of strings in Self-strings (S) does not match with solution from repertoire (P), then it is eliminated and replaced by the solution from repertoire (P). Negative Selection process for the Bees Algorithm can be summarised in two main steps as follows:

1) Step 1: matches a set of solution from repertoire (P) with strings in Self-strings. If there is no set of strings in Self-strings to match with (Self-strings is empty), then introduce the solution into Self-strings and repertoire (A).

2) Step 2: matches the next solution from repertoire (P) with strings in Self-strings. If no match is found, then add the solution into repertoire (A) and update string in Self-strings. Otherwise eliminate the solution. This step is repeated until repertoire (A) is full.

3.3.2 The enhanced Bees Algorithms

Two slightly different algorithms are proposed to solve single machine scheduling with a common due date.

3.3.2.1 The Bees Algorithm with Negative Selection: Single Swarm

The Bees Algorithm with Negative Selection is first developed to solve the benchmark when h value equals 0.2 and 0.4. In this case, the idle time does not need to be inserted. It means the optimal schedule can be found in a sequence that first job starts to be processed at time zero. Fig 3.7. shows its pseudo-code

The algorithm requires a number of parameters to be set, namely: number of scout bees (n), number of patches selected out of n visited points (m), number of best patches out of m selected patches (e), number of bees recruited for e best patches (nep), number of bees recruited for the other (m-e) selected patches (nsp), and the stopping criterion. The algorithm starts with the n scout bees being placed randomly in the search space (possible sequences).

In step 2, all jobs of each sequence are classified into two groups : early set and tardy set. Jobs finished early are in early set and jobs finished later than due date are in tardy set : $E_i = \max\{0, d_i. - C_i\}$ and $T_i = \max\{0, C_i - d_i\}$.

In step 3, all jobs of both sets are re-sequenced regarding v-shaped property: non-increasing order of the ratio p_j/α_j in early set and non-decreasing order of p_j/β_j in tardy set.

In step 4, the fitness values of the solutions visited by the scout bees are evaluated.

In steps 6 and 7, bees with the highest fitness values are chosen as "selected bees" and those sites that have been visited will be chosen for neighbourhood search. Then the algorithm conducts searches in the neighbourhood of the selected bees in terms of more bees for the e best bees. The latter can be chosen directly according to the fitness values associated with the sites they are visiting. In each search, one of four neighbourhood search operators is chosen randomly for each recruited bee. Chance to be chosen is given equally. After the search, the algorithm repeats steps 3 and 4 in order to calculate fitness values.

In steps 9 and 10, the process of negative selection then begins. The maximum number of best solutions that can be saved in the repertoire (A) is 5 percent of number of scout bees (n).

At the end of each generation, the colony will have new population from negative selection process and scout bees assigned to conduct random searches. Steps 4-11 are repeated until the best fitness value has stabilised. At the end of each generation, the colony will have two parts to its new population. The first part is the representative from previous generation and the second part is the new possible solutions conducted by other scout bees.

1. Initial population (sequences) with n random solutions.

2. Classify early and tardy jobs.

3. Re-sequence jobs in early and tardy sets regarding v-shaped property.

4. Evaluate fitness of the population.

5. While (stopping criterion has not been met).

6. Select sites (m) for neighbourhood search.

7. Recruit bees for selected sites: elite sites (e) and other selected sites (m-e).

8. Repeat step 3 and 4.

9. Move all solutions into repertoire (P) and sort them by their fitness values: high to low

10. Select the fittest bees by Negative Selection.

11. Assign remaining bees to search randomly and evaluate their fitness.

12. End while.

Figure 3.7 Pseudo-code of the Bees Algorithm: Single Swarm

3.3.2.2 The Bees Algorithm with Negative Selection: Two Swarms

This section presents the enhanced Bees Algorithm with Negative Selection that has the use of discrete uniform distribution technique and two swarms of bees to solve this single machine scheduling with a common due date benchmark when h value equals 0.6 and 0.8. To find an optimal solution, the idle time has to be inserted which means first job must not start at time zero.

To solve this dataset, a solution set is divided into two parts: continuous and combinatorial domains as shown in Fig. 3.8. Idle time is considered as continuous part. The pseudo-code of this algorithm is shown in Fig.3.9. During initialisation, the idle time is randomly generated by using discrete uniform distribution and inserted before the process of first job. In this version, the algorithm performs neighbourhood search for job sequence first and then performs idle time neighbourhood search after negative selection process. A group of recruited bees from a mini swarm is deployed in this process and the fittest bee will be selected from each site in step11. Then, in step 12, the remaining bees in the population are assigned randomly around the search space scouting for new potential solutions. These steps are repeated until a stopping criterion is met.

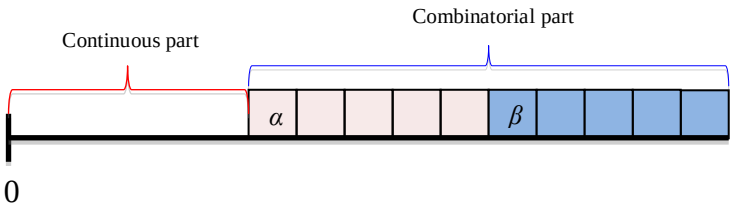


Figure 3.8 A set of solution with idle time considered as continuous part

1. Initial population (sequences) with n random solutions: an idle time follows by a set of sequence.

2. Classify early and tardy jobs.

3. Re-sequence jobs in early and tardy sets regarding v-shaped property.

4. Evaluate fitness of the population.

5. While (stopping criterion has not been met).

6. Select sites (m) for neighbourhood search.

7. Recruit bees for selected sites: elite sites (e) and other selected sites (m-e).

8. Repeat step 3 and 4.

9. Move all solutions into repertoire (P) and sort them by their fitness values: high to low

10. Select the fittest bees by Negative Selection.

11. Recruit bees from mini smarm for some best selected sites derived from negative selection to perform idle time neighbourhood search.

12. Repeat step 3 and 4.

13. Assign remaining bees to search for new solution randomly and evaluate their fitness.

14. End while.

Figure 3.9 Pseudo-code of the Bees Algorithm: Two swarms

3.4 Experimental results

The enhanced Bees Algorithms were implemented in Matlab, a high-level language, and run on Dell laptop: Intel (R) Core (TM)2 Duo CPU P8600 @2.40GHz, 4 GB RAM and MacBook Pro: Intel Quad Core i7 2.3GHz, 8 GB RAM. The algorithms have been applied to all 280 instances. Table 3.1 shows the parameters used for this experiment in search of potential solutions, where as Table 3.2 shows the parameters used in search of potential idle times.

Table 3.1 Parameters of Bees Algorithms

Parameters	**Value (when n = 10,20,50)**	**Value (when n = 100,200,500,1000)**
p : Population	200	500
m : Number of selected sites	50	75
e : Number of elites sites	20	30
nep : Number of bees around elite sites	30	30
nsp : Number of bees around other selected points	20	20

Table 3.2 Parameters used for idle time neighbourhood search

Parameters	**Value (when n = 10,20,50)**	**Value (when n = 100,200,500,1000)**
m : Number of selected sites	10	20
e : Number of elites sites	4	10
nep : Number of bees around elite sites	8	8
nsp : Number of bees around other selected points	4	4

The performance of the algorithms was quantified by the percentage of relative deviations (Δ) and standard deviation. To obtain the average performance of the algorithm, 10 runs were carried out for each instance to report the statistics based on the percentage of relative deviations from the upper bounds in Biskup and Feldmann (2001). To be more specific, Δ_{avg} was computed as follows:

$$\Delta_{avg} = \sum_{i=1}^{R} \left(\frac{(F_{eBA} - F_{ref})}{F_{ref}} \times 100 \right) / R \qquad \text{(Eq.3.4)}$$

where F_{eBA}, F_{ref} and R are the fitness function values generated by the enhanced Bees Algorithm in each run, the reference fitness function value generated by Feldmann and Biskup (2003), and the total number of runs, respectively. For convenience, $\Delta_{\min}$, $\Delta_{\max}$ and Δ_{std} denote the minimum, maximum and standard deviation of percentage of relative deviation in fitness function value over R runs, respectively.

Tables 3.3-3.16 illustrate the detail results of all seven categories of problems (when h = 0.2 and h = 0.4) obtained by Feldmann and Biskup (2003), Nearchou (2006), Pham et al. (2007b), and the enhanced Bees Algorithm with its Δ_{avg}.

Table 3.3 Computational results for 10 jobs when h = 0.2

n	instance	h = 0.2				
		UB	Cost DE	Basic BA	Enhanced BA	Δ_{avg}
10	1	1936	1936	1936	1936	0.00
	2	1042	1042	1042	1042	0.00
	3	1586	1586	1586	1586	0.00
	4	2139	2139	2139	2139	0.00
	5	1187	1187	1187	1187	0.00
	6	1521	1521	1521	1521	0.00
	7	2170	2170	2170	2170	0.00
	8	1720	1720	1720	1720	0.00
	9	1574	1574	1574	1574	0.00
	10	1869	1869	1869	1869	0.00

Table 3.4 Computational results for 10 jobs when h = 0.4

n	instance	h = 0.4				
		UB	Cost DE	Basic BA	Enhanced BA	Δ_{avg}
10	1	1025	1025	1025	1025	0.00
	2	615	615	615	615	0.00
	3	917	917	917	917	0.00
	4	1230	1230	1230	1230	0.00
	5	630	630	630	630	0.00
	6	908	908	908	908	0.00
	7	1374	1374	1374	1374	0.00
	8	1020	1020	1020	1020	0.00
	9	876	876	876	876	0.00
	10	1136	1136	1136	1136	0.00

Table 3.5 Computational results for 20 jobs when h = 0.2

n	instance	h = 0.2				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
20	1	4431	4394	4398	4394	-0.84
	2	8567	8430	8430	8430	-1.60
	3	6331	6210	6210	6210	-1.91
	4	9478	9188	9188	9188	-3.06
	5	4340	4215	4215	4215	-2.88
	6	6766	6527	6527	6527	-3.53
	7	11101	10455	10455	10455	-5.82
	8	4203	3920	3920	3920	-6.73
	9	3530	3465	3465	3465	-1.84
	10	5545	4979	4979	4979	-10.21

Table 3.6 Computational results for 20 jobs when h = 0.4

n	instance	h = 0.4				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
20	1	3066	3066	3067	3066	0.00
	2	4897	4847	4847	4847	-1.02
	3	3883	3838	3841	3838	-1.16
	4	5122	5118	5118	5118	-0.08
	5	2571	2495	2501	2495	-2.96
	6	3601	3582	3582	3582	-0.53
	7	6357	6238	6238	6238	-1.87
	8	2151	2145	2145	2145	-0.28
	9	2097	2096	2096	2096	-0.05
	10	3192	2925	2925	2925	-8.36

Table 3.7 Computational results for 50 jobs when h = 0.2

n	instance	h = 0.2				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
50	1	42363	40697	40704	40697	-3.93
	2	33637	30613	30613	30613	-8.99
	3	37641	34435	34425	34425	-8.54
	4	30166	27755	27760	27755	-7.99
	5	32604	32307	32307	32307	-0.91
	6	36920	34993	34970	34969	-5.28
	7	44277	43136	43136	43134	-2.58
	8	46065	43839	43840	43839	-4.83
	9	36397	34228	34228	34228	-5.96
	10	35797	32958	32961	32958	-7.93

Table 3.8 Computational results for 50 jobs when h = 0.4

n	instance	h = 0.4				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
50	1	24868	23792	23792	23792	-4.33
	2	19279	17910	17907	17907	-7.12
	3	21353	20500	20502	20500	-3.99
	4	17495	16657	16657	16657	-4.79
	5	18441	18007	18007	18007	-2.35
	6	21497	20385	20397	20385	-5.17
	7	23883	23038	23042	23038	-3.54
	8	25402	24888	24888	24888	-2.02
	9	21929	19984	19984	19984	-8.87
	10	20048	19167	19167	19167	-4.39

Table 3.9 Computational results for 100 jobs when h = 0.2

n	instance	h = 0.2				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
100	1	156103	145631	145516	145516	-6.78
	2	132605	124964	124916	124916	-5.80
	3	137463	129838	129800	129800	-5.57
	4	137265	129632	129584	129584	-5.60
	5	136761	124368	124351	124351	-9.07
	6	151938	139196	139193	139193	-8.39
	7	141613	135027	135026	135026	-4.65
	8	168086	160198	160147	160147	-4.72
	9	125153	116528	116522	116522	-6.90
	10	124446	118971	118913	118913	-4.45

Table 3.10 Computational results for 100 jobs when h = 0.4

n	instance	h = 0.4				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
100	1	89588	85897	85884	85884	-4.13
	2	74854	73002	72982	72981	-2.50
	3	85363	79690	79598	79598	-6.75
	4	87730	79405	79405	79405	-9.49
	5	76424	71334	71275	71275	-6.74
	6	86724	77789	77789	77789	-10.30
	7	79854	78250	78244	78244	-2.02
	8	95361	94365	94365	94365	-1.04
	9	73605	69527	69457	69457	-5.64
	10	72399	71951	71850	71850	-0.76

Table 3.11 Computational results for 200 jobs when h = 0.2

n	instance	h = 0.2				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
200	1	526666	498653	498653	498653	-5.32
	2	566643	541181	541180	541180	-4.49
	3	529919	488732	488665	488665	-7.78
	4	603709	586294	586257	586257	-2.89
	5	547953	513396	513217	513217	-6.34
	6	502276	478059	478019	478019	-4.83
	7	479651	454757	454757	454757	-5.19
	8	530896	494348	494276	494276	-6.90
	9	575353	529388	529275	529275	-8.01
	10	572866	538389	538332	538332	-6.03

Table 3.12 Computational results for 200 jobs when h = 0.4

n	instance	h = 0.4				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
200	1	301449	295767	295684	295684	-1.91
	2	335714	319212	319199	319199	-4.92
	3	308278	293980	293886	293888	-4.67
	4	360852	353113	353034	353034	-2.17
	5	322268	304666	304668	304666	-5.46
	6	292453	279982	279920	279920	-4.29
	7	279576	275095	275024	275024	-1.63
	8	288746	279323	279172	279172	-3.32
	9	331107	310558	310402	310402	-6.25
	10	332808	323325	323085	323085	-2.92

Table 3.13 Computational results for 500 jobs when h = 0.2

n	instance	h = 0.2				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
500	1	3113088	2954864	n/a	2954852	-5.08
	2	3569058	3365958	n/a	3365953	-5.69
	3	3300744	3103108	n/a	3103107	-5.99
	4	3408867	3221273	n/a	3221260	-5.50
	5	3377547	3114923	n/a	3114914	-7.78
	6	3024082	2792248	n/a	2792239	-7.67
	7	3381166	3172733	n/a	3172714	-6.17
	8	3376678	3122332	n/a	3122318	-7.53
	9	3617807	3364823	n/a	3364823	-6.99
	10	3315019	3120383	n/a	3120383	-5.87

Table 3.14 Computational results for 500 jobs when h = 0.4

n	instance	h = 0.4				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
500	1	1839902	1787906	n/a	1787899	-2.83
	2	2064998	1994930	n/a	1994915	-3.39
	3	1909304	1864827	n/a	1864685	-2.34
	4	1930829	1887781	n/a	1887604	-2.24
	5	1881221	1807272	n/a	1807251	-3.93
	6	1658411	1610343	n/a	1610188	-2.91
	7	1971176	1902962	n/a	1902833	-3.47
	8	1924191	1819358	n/a	1819355	-5.45
	9	2065647	1973837	n/a	1973780	-4.45
	10	1928579	1837530	n/a	1837485	-4.72

Table 3.15 Computational results for 1000 jobs when h = 0.2

n	instance	h = 0.2				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
1000	1	15190371	14056103	n/a	14055942	-7.47
	2	13356727	12296728	n/a	12296689	-7.94
	3	12919259	11974907	n/a	11974875	-7.31
	4	12705290	11805221	n/a	11805204	-7.08
	5	13276868	12457810	n/a	12457788	-6.17
	6	12236080	11653395	n/a	11653258	-4.76
	7	14160773	13286055	n/a	13286027	-6.18
	8	13314723	12279652	n/a	12279489	-7.78
	9	12433821	11764788	n/a	11764472	-5.38
	10	13395234	12433037	n/a	12433015	-7.18

Table 3.16 Computational results for 1000 jobs when h = 0.4

n	instance	h = 0.4				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
1000	1	8570154	8113004	n/a	8112904	-5.34
	2	7592040	7273409	n/a	7273368	-4.20
	3	7313736	6988905	n/a	6988904	-4.44
	4	7300217	7025750	n/a	7025544	-3.76
	5	7738367	7366803	n/a	7366619	-4.80
	6	7144491	6928294	n/a	6928077	-3.03
	7	8426024	7862538	n/a	7862431	-6.69
	8	7508507	7223809	n/a	7223732	-3.79
	9	7299271	7059399	n/a	7059358	-3.29
	10	7617658	7277199	n/a	7276948	-4.47

Tables 3.17-3.30 illustrate the detail results when h = 0.6 and h = 0.8 (with idle time inserted). Note that some of the results from the basic Bees Algorithm are not applicable.

The results obtained by the enhanced Bees Algorithm were compared with the results from Pham et al. (2007), Biskup and Feldmann (2001), Feldmann M, and Biskup D (2003), Hino et al. (2005), Pan et al. (2006), Nearchou (2006) and Talebi et al (2009). In Biskup and Feldmann (2001), the average percentage improvements and their standard deviations are given using the best solution among all the heuristics, namely, evolution search (ES), simulated annealing (SA), threshold accepting (TA) and TA with a back step (TAR). Since the enhanced Bees Algorithm is stochastic, its minimum, maximum, average and standard deviation of runs should be given to evaluate its performance. However, Hino et al. (2005) conducted 10 runs and selected the best out of 10 runs even updating the idle time. For this reason, the minimum percentage of relative deviation (Δ_{min}) of the enhanced Bees Algorithms was compared to Pham et al (2007), Hino et al. (2005) and Pan et al. (2006). Tables 3.31-3.34 summarise Δ_{min} of the computational results to be compared to Hino et al. (2005), Pan et al. (2006) and Pham et al (2007) with regard to h value respectively. As seen in Tables 3.31 and 3.32 when h = 0.2 and 0.4 there is not a large difference. In the average of all results when h = 0.2, the basic Bess Algorithm (BA) performed slightly better than the enhance Bees Algorithm but when h = 0.4 the enhanced Bees Algorithm performed vice versa. For h = 0.6 and h = 0.8, the enhanced Bees Algorithm outperformed other algorithms. See Tables 3.33-3.34, there is a great deal of

difference especially for 100 jobs. The enhanced Bees Algorithm, the BA, discrete particle swarm optimisation (DPSO) and GA have a similar tendency to yield negative percentage of relative deviations (Δ_{min}), which means they outperformed Biskup and Feldmann (2001). However, Tabu Search (TS), HTG (TS+GA) and HGT (GA+TS) show a tendency to diverge after 100 jobs and give positive percentage of relative deviations (Δ_{min}), which means they are inferior to Biskup and Feldmann (2001).

Table 3.17 Computational results for 10 jobs when h = 0.6

n	instance	h = 0.6				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
10	1	841	841	841	841	0.00
	2	615	615	615	615	0.00
	3	793	793	793	793	0.00
	4	815	815	815	815	0.00
	5	521	521	521	521	0.00
	6	755	755	755	755	0.00
	7	1101	1101	1101	1101	0.00
	8	610	610	610	610	0.00
	9	582	582	582	582	0.00
	10	710	710	710	710	0.00

Table 3.18 Computational results for 10 jobs when h = 0.8

n	instance	h = 0.8				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
10	1	818	818	818	818	0.00
	2	615	615	615	615	0.00
	3	793	793	793	793	0.00
	4	803	803	812	803	0.00
	5	521	521	521	521	0.00
	6	755	755	755	755	0.00
	7	1083	1083	1088	1083	0.00
	8	540	540	540	540	0.00
	9	554	554	554	554	0.00
	10	671	671	671	671	0.00

Table 3.19 Computational results for 20 jobs when h = 0.6

n	instance	h = 0.6				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
20	1	2986	2986	2987	2986	0.00
	2	3260	3206	3206	3206	-1.66
	3	3600	3583	3583	3583	-0.47
	4	3336	3317	3317	3317	-0.57
	5	2206	2173	2173	2173	-1.50
	6	3016	3010	3010	3010	-0.20
	7	4175	4126	4126	4126	-1.17
	8	1638	1638	1638	1638	0.00
	9	1992	1965	1965	1965	-1.36
	10	2116	2110	2116	2110	-0.28

Table 3.20 Computational results for 20 jobs when h = 0.8

n	instance	h = 0.8				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
20	1	2986	2986	2987	2986	0.00
	2	2980	2980	2980	2980	0.00
	3	3600	3583	3583	3583	-0.47
	4	3040	3040	3040	3040	0.00
	5	2206	2173	2173	2173	-1.50
	6	3016	3010	3010	3010	-0.20
	7	3900	3878	3878	3878	-0.56
	8	1638	1638	1638	1638	0.00
	9	1992	1965	1965	1965	-1.36
	10	1995	1995	1995	1995	0.00

Table 3.21 Computational results for 50 jobs when h = 0.6

n	instance	h = 0.6				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
50	1	17990	17969	17969	17969	-0.12
	2	14231	14050	14050	14050	-1.27
	3	16497	16497	16497	16497	0.00
	4	14105	14080	14080	14080	-0.18
	5	14650	14605	14605	14605	-0.31
	6	14251	14275	14251	14251	0.00
	7	17715	17616	17616	17617	-0.55
	8	21365	21329	21329	21329	-0.17
	9	14298	14202	14202	14202	-0.67
	10	14377	14366	14366	14366	-0.08

Table 3.22 Computational results for 50 jobs when h = 0.8

n	instance	h = 0.8				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
50	1	17990	17934	17934	17934	-0.31
	2	14132	14040	14040	14040	-0.65
	3	16497	16497	16497	16497	0.00
	4	14105	14080	14080	14081	-0.17
	5	14650	14605	14605	14605	-0.31
	6	14075	14066	14066	14066	-0.06
	7	17715	17616	17616	17616	-0.56
	8	21367	21335	21329	21329	-0.18
	9	13952	13948	13942	13942	-0.07
	10	14377	14363	14363	14363	-0.10

Table 3.23 Computational results for 100 jobs when h = 0.6

n	instance	h = 0.6				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
100	1	72019	72017	72017	71688	-0.46
	2	59351	59230	59230	59175	-0.30
	3	68537	68540	68537	68537	0.00
	4	69231	68774	68759	68759	-0.68
	5	55291	55345	55286	54887	-0.73
	6	62519	62411	62399	62278	-0.39
	7	62213	62204	62197	62187	-0.04
	8	80844	80713	80708	80351	-0.61
	9	58771	58730	58727	58729	-0.07
	10	61419	61366	61361	60966	-0.74

Table 3.24 Computational results for 100 jobs when h = 0.8

n	instance	h = 0.8				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
100	1	72019	72018	72017	71814	-0.28
	2	59351	59230	59230	59230	-0.20
	3	68537	68537	68537	68538	0.00
	4	69231	68772	68759	68760	-0.68
	5	55277	55103	55103	55103	-0.31
	6	62519	62407	62399	62399	-0.19
	7	62213	62197	62197	62197	-0.03
	8	80844	80713	80708	80713	-0.16
	9	58771	58727	58727	58466	-0.52
	10	61419	61361	61361	61341	-0.13

Table 3.25 Computational results for 200 jobs when h = 0.6

n	instance	h = 0.6				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
200	1	254268	255566	n/a	254259	0.00
	2	266028	267002	n/a	266002	-0.01
	3	254647	255337	n/a	254488	-0.06
	4	297269	298230	n/a	297109	-0.05
	5	260455	260981	n/a	260278	-0.07
	6	236160	236942	n/a	235702	-0.19
	7	247555	247450	n/a	246330	-0.49
	8	225572	226301	n/a	225215	-0.16
	9	255029	255519	n/a	254659	-0.15
	10	269236	268759	n/a	268353	-0.33

Table 3.26 Computational results for 200 jobs when h = 0.8

n	instance	h = 0.8				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
200	1	254268	255697	n/a	254259	0.00
	2	266028	267315	n/a	266002	-0.01
	3	254647	254911	n/a	254476	-0.07
	4	297269	297981	n/a	297109	-0.05
	5	260455	261458	n/a	260278	-0.07
	6	236160	236462	n/a	235702	-0.19
	7	247555	247450	n/a	246313	-0.50
	8	225572	225529	n/a	225215	-0.16
	9	255029	255675	n/a	254637	-0.15
	10	269236	269042	n/a	268354	-0.33

Table 3.27 Computational results for 500 jobs when h = 0.6

n	instance	h = 0.6				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
500	1	1581233	1617712	n/a	1579140	-0.13
	2	1715332	1741211	n/a	1712429	-0.17
	3	1644947	1680763	n/a	1641706	-0.20
	4	1640942	1684516	n/a	1640785	-0.01
	5	1468325	1477669	n/a	1468256	0.00
	6	1413345	1450456	n/a	1411867	-0.10
	7	1634912	1671889	n/a	1634330	-0.04
	8	1542090	1562208	n/a	1540458	-0.11
	9	1684055	1705411	n/a	1680486	-0.21
	10	1520515	1527515	n/a	1519215	-0.09

Table 3.28 Computational results for 500 jobs when h = 0.8

n	instance	h = 0.8				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
500	1	1581233	1610769	n/a	1579109	-0.13
	2	1715322	1733575	n/a	1712466	-0.17
	3	1644947	1653140	n/a	1641718	-0.20
	4	1640942	1653346	n/a	1640784	-0.01
	5	1468325	1481320	n/a	1468263	0.00
	6	1413345	1426017	n/a	1411841	-0.11
	7	1634912	1649639	n/a	1634330	-0.04
	8	1542090	1560903	n/a	1540470	-0.11
	9	1684055	1707100	n/a	1680647	-0.20
	10	1520515	1529451	n/a	1519205	-0.09

Table 3.29 Computational results for 1000 jobs when h = 0.6

n	instance	h = 0.6				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
1000	1	6411581	6421773	n/a	6411260	-0.01
	2	6112598	6158588	n/a	6110369	-0.04
	3	5985538	6078028	n/a	5983589	-0.03
	4	6096729	6198005	n/a	6088472	-0.14
	5	6348242	6448069	n/a	6342433	-0.09
	6	6082142	6230516	n/a	6079207	-0.05
	7	6575879	6608387	n/a	6574569	-0.02
	8	6069658	6153974	n/a	6067688	-0.03
	9	6188416	6280472	n/a	6185834	-0.04
	10	6147295	6230598	n/a	6146054	-0.02

Table 3.30 Computational results for 1000 jobs when h = 0.8

n	instance	h = 0.8				
		UB	**Cost DE**	**Basic BA**	**Enhanced BA**	Δ_{avg}
1000	1	6411581	6611622	n/a	6411352	0.00
	2	6112598	6365048	n/a	6110400	-0.04
	3	5985538	6077715	n/a	5983430	-0.04
	4	6096729	6239392	n/a	6089268	-0.12
	5	6348242	6488538	n/a	6342525	-0.09
	6	6082142	6321170	n/a	6079243	-0.05
	7	6575879	6717260	n/a	6574465	-0.02
	8	6069658	6155240	n/a	6067727	-0.03
	9	6188416	6434096	n/a	6185813	-0.04
	10	6147295	6337246	n/a	6145999	-0.02

Table 3.31 Comparison of minimum deviation of computational results: h = 0.2

n	h = 0.2						
	DPSO	**TS**	**GA**	**HTG**	**HGT**	**Basic BA**	**Enhanced BA**
10	0.00	0.25	0.12	0.12	0.12	0.00	0.00
20	-3.84	-3.84	-3.84	-3.84	-3.84	-3.84	-3.84
50	-5.70	-5.70	-5.68	-5.70	-5.70	-5.70	-5.70
100	-6.19	-6.19	-6.17	-6.19	-6.19	-6.19	-6.19
200	-5.78	-5.76	-5.74	-5.76	-5.76	-5.78	-5.78
500	-6.42	-6.41	-6.41	-6.41	-6.41	-6.43	-6.43
1,000	-6.76	-6.73	-6.75	-6.74	-6.74	**-6.76**	-6.72
AVG	-4.96	-4.91	-4.92	-4.93	-4.93	**-4.96**	-4.95

Table 3.32 Comparison of minimum deviation of computational results: h = 0.4

n	h = 0.4						
	DPSO	**TS**	**GA**	**HTG**	**HGT**	**Basic BA**	**Enhanced BA**
10	0.00	0.24	0.19	0.19	0.19	0.00	0.00
20	-1.63	-1.62	-1.62	-1.62	-1.62	-1.63	-1.63
50	-4.66	-4.66	-4.60	-4.66	-4.66	-4.66	-4.66
100	-4.94	-4.93	-4.91	-4.93	-4.93	-4.94	-4.94
200	-3.75	-3.74	-3.75	-3.75	-3.75	-3.75	-3.75
500	-3.56	-3.57	-3.58	-3.58	-3.58	-3.57	-3.57
1,000	-4.37	-4.39	-4.40	-4.39	-4.39	-4.35	**-4.38**
AVG	-3.27	-3.24	-3.24	-3.25	-3.25	-3.27	**-3.28**

Table 3.33 Comparison of minimum deviation of computational results: h = 0.6

n	h = 0.6						
	DPSO	**TS**	**GA**	**HTG**	**HGT**	**Basic BA**	**Enhanced BA**
10	0.00	0.10	0.03	0.03	0.01	0.00	0.00
20	-0.72	-0.71	-0.68	-0.71	-0.71	-0.72	-0.72
50	-0.34	-0.32	-0.31	-0.27	-0.31	**-0.34**	-0.33
100	-0.15	-0.01	-0.12	0.08	0.04	-0.15	**-0.40**
200	-0.15	-0.01	-0.13	0.37	0.07	-0.15	-0.15
500	-0.11	0.25	-0.11	0.73	0.15	-0.11	-0.11
1,000	-0.06	1.01	-0.05	1.28	0.42	-0.05	0.05
AVG	-0.22	0.04	-0.20	0.22	-0.05	-0.22	**-0.24**

Table 3.34 Comparison of minimum deviation of computational results: h = 0.8

n	h = 0.8						
	DPSO	**TS**	**GA**	**HTG**	**HGT**	**Basic BA**	**Enhanced BA**
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	-0.41	-0.41	-0.28	-0.41	-0.41	-0.41	-0.41
50	-0.24	-0.24	-0.19	-0.23	-0.23	-0.24	-0.24
100	-0.18	-0.15	-0.12	-0.08	-0.11	-0.18	**-0.25**
200	-0.15	-0.14	-0.14	0.26	0.07	-0.15	-0.15
500	-0.11	0.21	-0.11	0.73	0.13	-0.11	-0.10
1,000	-0.06	1.13	-0.05	1.28	0.40	-0.05	-0.05
AVG	-0.16	0.07	-0.13	0.22	-0.02	-0.16	**-0.17**

Table 3.35 shows comparative results for the Enhanced Bees Algorithm, BA and DPSO in terms of minimum, maximum and average percentage of relative deviations and standard deviations. The minimum percentage of relative deviations ($\Delta_{\min}$) of the enhanced Bees Algorithm was compared to the Scatter Search Algorithm (SS) (Talebi et al. 2009), the BA, and the DPSO. The average percentage of relative deviation (Δ_{avg}) of the enhanced Bees Algorithm was compared to the BA, the DPSO and differential evolution (DE). It was found that the enhanced Bees Algorithm outperforms these four algorithms. It can be seen from the total minimum, that the enhanced Bees Algorithm is slightly better than the BA and the DPSO at -2.15 and much better than the SS at 2.15, which is inferior to Biskup and Feldmann (2001).

For 100 jobs when h = 0.6 or 0.8, the enhanced Bees Algorithm is superior to the BA and DPSO which can perform better than the DE. As can be seen, the standard deviation for both the enhanced Bees Algorithm are nearly zero, which means that it is slightly more robust than DPSO. All the statistics obtained show that the performance of the enhanced Bees Algorithm is better than the basic BA and is superior to all existing approaches considered in this study.

In term of runtime, the stopping criteria of the BA is 1,000 iterations or 2,000 iterations in some difficult instances whereas the stopping criteria of the enhanced Bees Algorithm is set to stop when the solution found was less than or equal to the upper bound or it is reached 1,000 iterations. In many cases especially when

solving 10 jobs, the enhanced Bees Algorithm found the optimum after performing not more than 10 or 20 iterations.

Table 3.35 Comparison between the enhance Bees Algorithms, the basic Bees Algorithm, DPSO and DE

n	h	D_{min}				D_{max}			Δ_{avg}				D_{std}		
		SS	**DPSO**	**BA**	**eBA**	**DPSO**	**BA**	**eBA**	**DPSO**	**DE**	**BA**	**eBA**	**DPSO**	**BA**	**eBA**
10	**0.2**	0.33	0.00	0.00	0.00	0.11	0.00	0.00	0.01	0.00	0.00	0.00	0.03	0.00	0.00
	0.4	0.19	0.00	0.00	0.00	0.15	0.00	0.00	0.02	0.00	0.00	0.00	0.05	0.00	0.00
	0.6	1.54	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.8	0.70	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	**0.2**	-3.57	-3.84	-3.84	-3.84	-3.79	-3.83	-3.84	-3.83	-3.84	-3.84	-3.84	0.02	0.00	0.00
	0.4	-0.85	-1.63	-1.63	-1.63	-1.57	-1.63	-1.63	-1.62	-1.63	-1.63	-1.63	0.02	0.00	0.00
	0.6	-2.9	-0.72	-0.72	-0.72	-0.66	-0.72	-0.72	-0.71	-0.72	-0.72	-0.72	0.03	0.00	0.00
	0.8	-6.82	-0.41	-0.41	-0.41	-0.41	-0.41	-0.41	-0.41	-0.41	-0.41	-0.41	0.00	0.00	0.00
50	**0.2**	-5.23	-5.70	-5.70	-5.70	-5.61	-5.69	-5.70	-5.68	-5.69	-5.70	-5.70	0.03	0.00	0.00
	0.4	-4.05	-4.66	-4.66	-4.66	-4.52	-4.66	-4.66	-4.63	-4.66	-4.66	-4.66	0.05	0.00	0.00
	0.6	-1.62	-0.34	-0.34	-0.33	-0.23	-0.34	-0.33	-0.31	-0.32	-0.34	-0.33	0.04	0.00	0.00
	0.8	-3.13	-0.24	-0.24	-0.24	-0.24	-0.22	-0.24	-0.24	-0.24	-0.24	-0.24	0.00	0.01	0.00

Table 3.35 Comparison between the enhance Bees Algorithms, the basic Bees Algorithm, DPSO and DE (continued)

n	h	D_{min}				D_{max}			Δ_{avg}				D_{std}		
		SS	**DPSO**	**BA**	**eBA**	**DPSO**	**BA**	**eBA**	**DPSO**	**DE**	**BA**	**eBA**	**DPSO**	**BA**	**eBA**
100	**0.2**	-5.82	-6.19	-6.19	-6.19	-6.15	-6.19	-6.19	-6.18	-6.17	-6.19	-6.19	0.02	0.00	0.00
	0.4	-4.28	-4.94	-4.94	-4.94	-4.82	-4.93	-4.94	-4.90	-4.89	-4.94	-4.94	0.04	0.00	0.00
	0.6	-0.27	-0.15	-0.15	-0.40	0.26	-0.14	-0.40	-0.09	-0.13	-0.14	-0.40	0.14	0.00	0.00
	0.8	0.37	-0.18	-0.18	-0.25	-0.18	-0.17	-0.25	-0.18	-0.17	-0.18	-0.25	0.00	0.00	0.00
200	**0.2**	-5.37	-5.78	-5.78	-5.78	-5.74	-5.77	-5.78	-5.77	-5.77	-5.78	-5.78	0.01	0.00	0.00
	0.4	-3.12	-3.75	-3.75	-3.75	-3.68	-3.74	-3.75	-3.72	-3.72	-3.75	-3.75	0.02	0.01	0.00
	0.6	0.19	-0.15	-0.15	-0.15	0.56	-0.15	-0.15	-0.03	0.23	-0.15	-0.15	0.27	0.00	0.00
	0.8	0.43	-0.15	-0.15	-0.15	-0.15	-0.15	-0.15	-0.15	0.20	-0.15	-0.15	0.00	0.00	0.00
500	**0.2**	-5.93	-6.42	-6.43	-6.43	-6.40	-6.42	-6.43	-6.41	-6.43	-6.43	-6.43	0.01	0.00	0.00
	0.4	-3.06	-3.56	-3.57	-3.57	-3.51	-3.56	-3.57	-3.54	-3.57	-3.57	-3.57	0.01	0.00	0.00
	0.6	0.31	-0.11	-0.11	-0.11	-0.11	-0.11	-0.11	-0.11	1.72	-0.11	-0.11	0.00	0.00	0.00
	0.8	0.38	-0.11	-0.11	-0.10	-0.11	-0.11	-0.10	-0.11	1.01	-0.11	-0.10	0.00	0.00	0.00

Table 3.35 Comparison between the enhance Bees Algorithms, the basic Bees Algorithm, DPSO and DE (continued)

n	h	D_{min}				D_{max}			Δ_{avg}				D_{std}		
		SS	**DPSO**	**BA**	**eBA**	**DPSO**	**BA**	**eBA**	**DPSO**	**DE**	**BA**	**eBA**	**DPSO**	**BA**	**eBA**
1000	**0.2**	-6.18	-6.76	-6.76	-6.72	-6.73	-6.74	-6.72	-6.75	-6.75	-6.72	-6.72	0.01	0.01	0.00
	0.4	-3.76	-4.37	-4.35	-4.38	-4.32	-4.33	-4.38	-4.35	-4.34	-4.38	-4.38	0.01	0.01	0.00
	0.6	0.71	-0.06	-0.05	-0.05	-0.03	-0.05	-0.04	-0.04	1.29	-0.05	-0.05	0.01	0.01	0.01
	0.8	0.71	-0.06	-0.05	-0.05	-0.06	-0.05	-0.04	-0.06	2.79	-0.05	-0.05	0.00	0.00	0.01
AVG		2.15	-2.15	-2.15	-2.16	-2.07	-2.15	-2.16	-2.14	-1.87	-2.15	-2.16	0.03	0.00	0.00

3.5 Summary

In this chapter, the enhanced Bees Algorithm is proposed. The aim is to improve the basic Bees Algorithm's performance in solving single machine with common due date problem. Negative Selection is embedded into the basic Bees Algorithm to overcome its drawback. The results are compared to those obtained by the basic Bees Algorithm and by some other well-known algorithms to be found in the literatures. The results obtained show that the enhanced Bees Algorithm performs better than the basic version and any other well-known algorithms considered for this problem.

CHAPTER 4

THE BACTERIAL BEES ALGORITHM TO MINIMISE TOTAL WEIGHTED TARDINESS ON A MACHINE SCHEDULING

4.1 Preliminaries

Single machine total weighted tardiness problem is one of the well-known scheduling problems. It is known to be NP-hard (Lenstra et al. 1977) that consists of one machine and a number of independent jobs. The objective of this scheduling problem is to find a processing order of all jobs that minimise the sum of the weighted tardiness. In the first phase of this research, the Bees Algorithm with four different neighbourhood search procedures has been developed. It could solve 122 out of 125 instances of 40 job problem benchmark (Pham et al 2012).

However, it struggled to find optimal solutions of many instances of 50 and 100 job problems. In the second phase of this research, bacterial foraging technique was adapted and embedded into the Bees Algorithm to improve its performance.

The chapter is organised as follows: Section 4.2 describes single machine total weighted tardiness problem and benchmark used in this study. Section 4.3 presents the Bacterial Bees Algorithm developed to solve this problem. Its characteristics are described. In section 4.4, results are compared with the results derived from the first Bees Algorithm developed to solve this benchmark and other existing works to show the improvement. The summary of this work is in Section 4.5.

4.2 Single Machine Total Weighted Tardiness Problem

The single machine total weighted tardiness problem is to schedule n jobs on a machine. A set of jobs is to be processed without interruption on a machine that can handle one job at a time. Each job i is available for processing at time zero and has a processing time p_i, a weight w_i, and a due date d_i by which it should ideally be finished. The tardiness of a job i can be defined as $T_i = \max \{0, C_i - d_i\}$, where C_i is the completion time of job i. The objective of this scheduling problem is to find a processing order of all jobs that minimise the sum of the weighted tardiness given by :

$$\sum_{i=1}^{n} w_i T_i \qquad \text{(Eq.4.1)}$$

The benchmark data used in this study can be obtained at OR-LIBRARY http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html. 125 test instances are available for each problem size n = 40, n = 50 and n = 100 where n is number of jobs. The instances were randomly generated as follows:

- For each job i (i = 1 ,..., n), an integer processing time p_i was generated from the uniform distribution (1,100) and integer processing weight w_i was generated from the uniform distribution (1,10). Instance classes of varying hardness, the due dates, were generated by using different uniform distributions.

- For a given relative range of due dates (RDD) = 0.2, 0.4, 0.6, 0.8, 1.0 and a given average tardiness factor (TF) = 0.2, 0.4, 0.6,0.8,1.0, an integer due date d_i for job i was randomly generated from the uniform distribution [P(1-TF-RDD/2), P(1-TF+RDD/2)], where

$$P = \sum_i^n p_i \qquad \text{(Eq.4.2)}$$

The optimal values for 40 and 50 job problems and best-known optimal values for 100 job problem are known and also available at OR-LIBRARY. Those optimal values of 40 and 50 job problems are from Crauwels et al. (1996) and of 100 job problem is from Congram et al. (1998). Table 4.1 shows optimal values for 40 and 50 job problems, and best-known for 100 job problems respectively.

Table 4.1 Optimal and Best-known solutions of 40, 50, and 100 job problems

Instance	Optimum for 40 jobs	Optimum for 50 jobs	Best-known for100 jobs
1	913	2134	5988
2	1225	1996	6170
3	537	2583	4267
4	2094	2691	5011
5	990	1518	5283
6	6955	26276	58258
7	6324	11403	50972
8	6865	8499	59434
9	16225	9884	40978
10	9737	10655	53208
11	17465	43504	181649
12	19312	36378	234179
13	29256	45383	178840
14	14377	51785	157476
15	26914	38934	172995
16	72317	87902	407703
17	78623	84260	332804
18	74310	104795	544838
19	77122	89299	477684
20	63229	72316	406094
21	77774	214546	898925
22	100484	150800	556873
23	135618	224025	539716
24	119947	116015	744287
25	128747	240179	585306

Table 4.1 Best-known solution values of 40, 50, and 100 job problem (continued)

Instance	40 jobs	50 jobs	100 jobs
26	108	2	8
27	64	4	718
28	15	755	27
29	47	99	480
30	98	22	50
31	6575	9934	24202
32	4098	7178	25469
33	5468	4674	32964
34	2648	4017	22215
35	5290	6459	19114
36	19732	34892	108293
37	17349	22739	181850
38	24499	29467	90440
39	19008	49352	151701
40	19611	26423	129728
41	57640	71111	462324
42	81462	90163	425875
43	65134	84126	320537
44	78139	123893	360193
45	66579	79883	306040
46	64451	157505	829828
47	113999	133289	623356
48	74323	191099	748988
49	110295	150279	656693
50	95616	198076	599269

Table 4.1 Best-known solution values of 40, 50, and 100 job problem (continued)

Instance	40 jobs	50 jobs	100 jobs
51	0	0	0
52	0	0	0
53	0	0	0
54	0	0	0
55	0	0	0
56	2099	1258	9046
57	2260	3679	11539
58	4936	2522	16313
59	3784	3770	7965
60	3289	5904	19912
61	20281	25212	86793
62	13403	17337	87067
63	19771	30729	96563
64	24346	18082	100788
65	14905	25028	56510
66	65386	76878	243872
67	65756	85413	401023
68	78451	92756	399085
69	81627	77930	309232
70	68242	74750	222684
71	90486	150580	640816
72	115249	131680	611362
73	68529	98494	623429
74	79006	135394	584628
75	98110	135677	575274

Table 4.1 Best-known solution values of 40, 50, and 100 job problem (continued)

Instance	40 jobs	50 jobs	100 jobs
76	0	0	0
77	0	0	0
78	0	0	0
79	0	0	0
80	0	0	0
81	684	816	1400
82	172	4879	317
83	798	973	1146
84	617	508	136
85	776	3780	284
86	10262	20751	66850
87	18646	36053	84229
88	10021	28268	55544
89	25881	28846	54612
90	8159	15451	75061
91	47683	89298	248699
92	43004	66340	311022
93	55730	61060	326258
94	59494	42453	273993
95	42688	56522	316870
96	126048	177909	495516
97	114686	139591	636903
98	112102	148906	680082
99	98206	179264	622464
100	157296	120108	449545

Table 4.1 Best-known solution values of 40, 50, and 100 job problem (continued)

Instance	40 jobs	50 jobs	100 jobs
101	0	0	0
102	0	0	0
103	0	0	0
104	0	0	0
105	0	0	0
106	0	0	0
107	516	1717	1193
108	3354	0	0
109	0	6185	232
110	0	1295	0
111	31478	27310	159138
112	21169	15867	174377
113	27077	35106	91171
114	19648	15467	168297
115	13774	10574	70190
116	46770	35727	370631
117	50364	71922	324437
118	25460	65433	246243
119	66707	106043	293576
120	69019	101665	267326
121	122266	78315	471214
122	82456	119925	570459
123	75118	101157	397029
124	73041	139488	431115
125	104531	110392	560754

4.3 The Bacterial Bees Algorithm for Single Machine Total Weighted Tardiness Problem

The Bees Algorithm was successfully developed to solve 40 job problem (Pham et al 2012). In this basic version, four different neighbourhood search procedures were deployed randomly. Figures 4.1-4.4 display neighbourhood search procedures deployed for the Bees Algorithm. Figure 4.1 shows 1st procedure: swap between two jobs selected randomly. Two pairs of jobs will be done in this process. Figure 4.2 shows 2nd procedure: reverse job order in a selected sub sequence. Two positions are selected randomly then job positions between these two positions are reversed. Figure 4.3 shows 3rd procedure: swap between two sub sequences. Position one and two are selected randomly first and then position three and four. Provided these selections do not overlap then job sequence between position one and two and job sequence between position three and four are swapped. Figure 4.4 shows 4th procedure: swap between three jobs. Three positions are selected and then swapped. The job at selected position one will be moved to selected position two, the previous job at selected position two will be moved to position three, and the job at position three will be moved to selected position one.

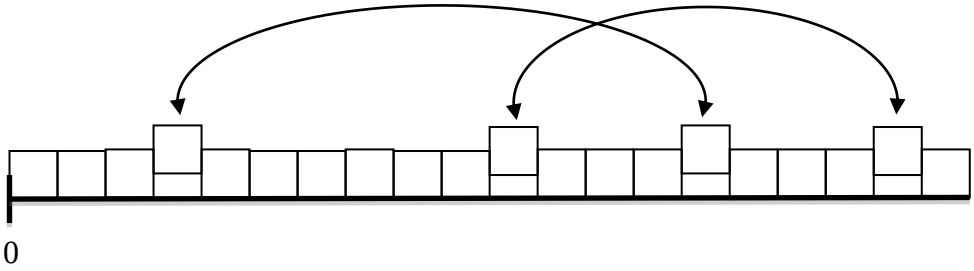
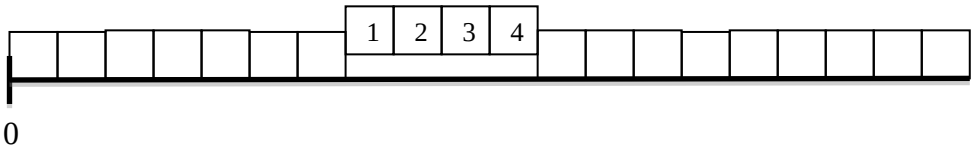
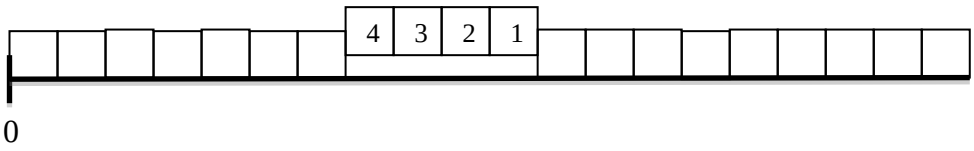


Figure 4.1 Swapping between two jobs randomly



a) before



b) after

Figure 4.2 Reversing job order in a selected sub sequence.

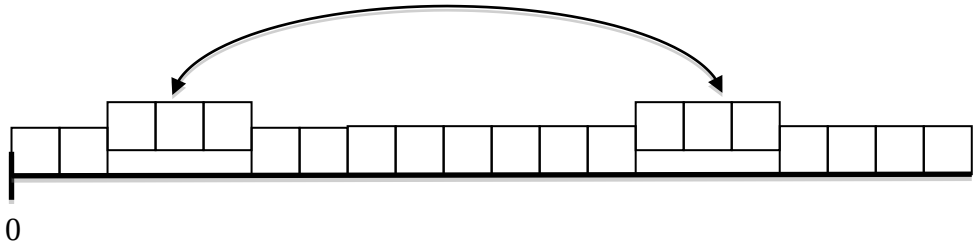


Figure 4.3 Swapping two groups of jobs

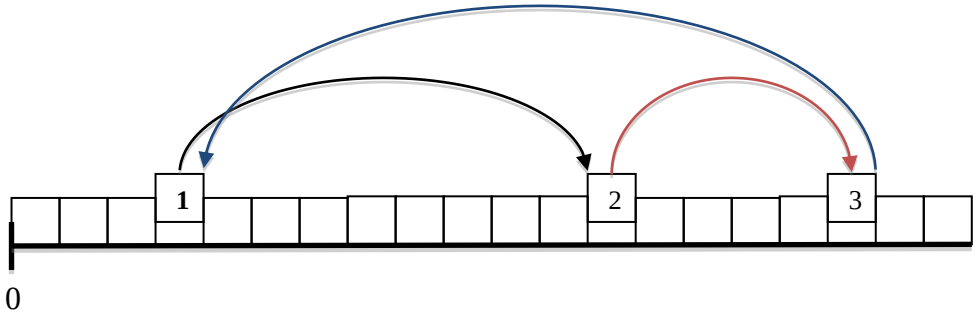


Figure 4.4 Swapping three job positions

The experimental results showed that the Bees Algorithm could find 122 optimal values out of 125 instances for 40 job problem. The full result details will be shown and discussed in Section 4.3. The result showed that it was struggling to find optimal values of many instances of 50 and 100 job problems. Hence the Bacterial foraging technique is considered for enhancing the Bees Algorithm's performance in this task.

In 2002, Passino proposed Bacterial Foraging Optimisation Algorithm (BFOA) for Distributed Optimisation and Control. Foraging behaviour of E. coli, which is a common type of bacteria living in human intestine, was considered. BFOA consists of three events namely chemotaxis, reproduction, and elimination and disposal. The idea is to find the minimum of $J(\theta)$ where θ is the position of a bacterium and $J(\theta)$ represents the combine affects of attractants and repellents from environment. $J(\theta) < 0$, $J(\theta) = 0$, and $J(\theta) > 0$ represent that the bacterium at location θ is in nutrient-rich, neutral, and noxious environments, respectively. Basically, chemotaxis is a foraging behaviour that implements a type of optimisation where bacteria try to climb up the nutrient concentration and avoid noxious substances. It implements a type of biased random walk. Normally, each bacterium can move in two different ways. It can swim for a period of time in the same direction, or it may tumble, and alternate between these two modes of operation for the entire lifetime (Zhong et al 2011).

After a period of food search, the foraging strategies of some bacteria appear inferior evidently. To avoid noxious substances, last half of bacteria with high

cost are considered unhealthy and removed out of population. Healthiest bacteria each split into two bacteria to keep the population size constant. It is also possible that the local environment where a population of bacteria live changes either gradually or suddenly due to some other influence. For example, the sudden increase of temperature can kill a population of bacteria that are currently in a region with a high concentration of nutrients. This event is called elimination and dispersal and it is triggered with probability. If a certain individual satisfies the dispersal condition, it should be deleted and a new individual should be generated.

Chemotaxis and elimination and dispersal have been adapted and embedded into the Bees Algorithm in this research. These two techniques are re-designed to suit the Bees Algorithm for combinatorial optimisation problem. This new version is called 'the Bacterial Bees Algorithm'. The pseudo-code of the Bacterial Bees Algorithm is given in Fig 4.5. This algorithm starts with the bee foraging part. Initial population of *n* scout bees are randomly generated. Each bee presents a sequence of jobs. In step 2, the fitness computation process is carried out. In step 4, the *m* sites with highest fitness are selected for neighbourhood search. In step 5, the algorithm conducts searches around the selected site, assigning more bees to search in the surrounding area of the best '*e'* sites. Each bee randomly chooses to perform one of four neighbourhood search procedures. The fitness values are evaluated. For the first iteration, the fittest value is saved. In other iteration, the best fitness solution is compared with the saved one. If its value is less than the saved one, then overwrite the value and update $J(\theta)$. In this case $J(\theta) = 0$ which means $J(\theta)$ is in nutrient rich environment. In step 7, a small number of best

solutions (*s*) will be carried out for next iteration. Then only half of *n* including *s* is produced as new population for next generation. This is to reduce computational time from calculating fitness values. The Algorithm will repeat steps 2 to 6 until the best fitness is equal to the saved one which means $J(\theta) > 0$ and is in neutral environment, and then goto step 8. When this happens, to avoid local optimum the algorithm will do reproduction by keeping small number of best solution and clone them and move first or last job and insert it into a new random position to create new solution and then repeat steps 2 to 6 but in step 5, some of recruited bees will randomly perform chemotactic step. This study has adapted chemotactic step to a neighbourhood search procedure. The recruited bee will randomly choose and perform two neighbourhood search procedures with probability 0.25. If the algorithm could not improve the fitness value in a certain time, the best solution is save and the algorithm will perform elimination and dispersal event by ignoring all solutions and reproducing *n*-1 possible solutions in step 9. Then repeat steps 2-6 without chemotactic step.

1. Initial population (θ) with n random solutions.

2. Evaluate fitness of the population.

3. While (stopping criterion has not been met).

4. Select sites (m) for neighbourhood search.

5. Recruit bees for selected sites: elite sites (e) and other selected sites (m-e).

6. Evaluate the fittest values and for 1st iteration, save the best fittest (F_i) otherwise update $J(\theta)$ as follows:

If $F_i < F_{i-1}$

Then $J(\theta)$ is in nutrient rich environtment. Update best fitness and go to step (7)

Else if $F_i = F_{i-1}$ and less than T times

Then $J(\theta)$ is in neutral environtment. Go to step (8)

Else go to step (9).

7. Keep a small number of best solutions (s) and assign remaining bees to search for new possible solutions (p) where $p = (n/2)$-s. Then repeat step (2)-(6).

8. Keep a small number of best solutions, perform reproduction, and assign remaining bees to search for new possible solution (np) where $np = (n/2)$-$2s$. Then repeat step (2)-(6) with Chemotactic event in step (5).

9. Save the best solution (F) and perform elimination and dispersal event. Then reproduce n-1 possible solutions randomly.

10. End while.

Figure 4.5 The pseudo-code of the Bacterial Bees Algorithm

4.4 Experimental results

The first Bees Algorithm was implemented in Matlab and run on a cluster called Merlin provided by ARCCA, Cardiff University. The configuration and features for compute nodes are Xeon E5472 3.0GHz, 1600MHz FSB, 16 GB RAM, 12 MB L2 cache, 160 GB@ 7.2k RPM SATA HDD local disk. For the computation results, the Bees Algorithm was able to find 122 optimal solutions out of 125 instances. Three instances where the Bees Algorithm could not find the optimums are instance 62, 85, and 112.

The Bacterial Bees Algorithm is introduced to increase the performance. The 10 time experiment has been carried out on Dell laptop: Intel (R) Core (TM) 2 Duo CPU P8600 @2.40GHz 4 GB RAM and MacBook Pro: Intel Quad Core i7 2.3GHz. 6 GB RAM. This enhanced version of the Bees Algorithm found 125 optimal solutions out of 125 instances. Table 4.2 shows the parameters used and for the maximum number of being in neutral medium or trapping in local optima is normally set to 3 but only at some difficult instances, this parameter was set to 6 or 9. Table 4.3-4.7 show the comparison of running times between the basic Bees Algorithm and the Bacterial Bees Algorithm. The performance of the new algorithm was quantified by the average percentage of relative deviations (Δ_{avg}) from 10 runs. The minimum of the average percentage of relative deviations of computational runtime is -50.37. It performed 2 times faster than the basic Bees Algorithm on 13^{th} instance. The maximum of the average percentage of relative

deviations of computational time is -99.80. It found optimum 495 times faster than the basic Bees Algorithm on 87th instance.

Table 4.8-4.10 show the computational time of the Bacterial Bees Algorithm for 50 jobs problem. It could find 120 optimal solutions out of 125 instances. The minimum average runtime that it could find the optimum is 0.70 seconds on 103rd instance, whereas the maximum average runtime is at 1228.14 seconds on 107th instance. Table 4.11-1.13 show the computational time of the Bacterial Bees Algorithm for 100 job problem. It could find only 98 optimal solutions out of 125 instances. The minimum average runtime that it could find the optimum is 2.49 seconds on 77th instance, whereas the maximum average runtime is at 63427.00 seconds on 100th instance.

There are many existing research that had applied varied techniques to solve this benchmark. However, this study could not show and compare the results from the Bacterial Bees Algorithm with those existing work in detail as none of them reported or has showed results in detail. Some works used only some instances of each dataset problem to be tested on their techniques. Some works used only one or two dataset problems for their research.

Nearchou (2004) has applied a Simulated Annealing Algorithm on 40 and 50 jobs problems. 5 run were carried out. This algorithm solved to optimality 91 out of test instances for 40 jobs problem and 73 out of 125 test instance for 50 job problem. In 2006, Huang and Tung have introduced Discrete Particle Swarm Optimisation called MPSO to solve the two dataset. However, only 10 instances of each dataset were used to test the algorithm. Those instances are 1, 11, 21, 31,

46, 56, 71, 91,101, and 116. MPSO could solve all 10 instances of both dataset. Its performance was compared with previous work by Cagnina et al (2004) who applied a hybrid PSO which could solve only 5 out of 10 instances for 40 jobs problem and 4 out of 10 instances.

In 2006, Ferrolho and Crisostomo proposed Genetic Algorithm to solve some of test instances of 40, 50 and 100 jobs problems. For 40 jobs problem, 2^{nd}, 7^{th}, and 31^{st} instances were used. The average runtimes were 190.00, 362.40, 319.70 seconds respectively where as the Bacterial Bees Algorithm's average runtimes for these instances were 16.35, 8.48, and 14.78 seconds respectively. For 50 jobs problem, 1^{st} , 30^{th}, and 33^{rd} instances were used. The average runtimes were 88.30, 45.50, 573.60 seconds respectively where as the Bacterial Bees Algorithm's average runtimes for these instances were 4.42, 3.84, and 28.25 seconds respectively. For 100 jobs problem, 1^{st}, 4^{th} , and 26^{th} instances were used. The average runtimes were 2406.01, 2428.10, 523.90 seconds respectively where as the Bacterial Bees Algorithm's average runtimes for these instances were 64.10, 31.33, and 50.56 seconds respectively. However, both techniques were tested on different types of computers.

Kellegoz et al. (2008) selected first five instances of each job problem to compare the performances of 11 crossover operators to solve this total weighted tardiness problem. For 40 jobs problem, none of 11 crossover operators could find the optimums of 1^{st} – 4^{th} instances. Only 4 operators could find the optimum of 5^{th} instance. For 50 and 100 jobs problems, none of those operators could find the optimums of selected instances.

Tasgetiren et al (2004) proposed PSO_{spv} to solve this benchmark. A heuristic rule called Smallest Position Value (SPV) rule was developed to enable PSO to solve this combinatorial problem. PSO_{spv} was able to find 120 optimal solution out of 125 instances for 40 jobs problem, 110 optimal solution out of 125 instances for 50 jobs problem, and 51 best known solutions out of 125 instances for 100 jobs problem which is the most difficult one. It seems that the basic Bees Algorithm performed better in solving 40 jobs problem and Bacterial Bees Algorithm could perform better in all problems. However, this work set has limited runtime to 5 seconds for 40 jobs problem, 10 seconds for 50 jobs problem, and 100 seconds for 100 jobs problem. In term of runtime comparison, PSO_{spv} performed better than the Bees Algorithms.

In 2000, Besten et al presented the Ant Colony Optimisation (ACO) to solve this benchmark. Their results are superior to the Bacterial Bees Algorithm's. The ACO found all optimal solutions for 40 and 50 jobs problems and found 113 out of 125 instances for 100 jobs problem.

Table 4.2 Parameters of the Bacterial Bees Algorithms

Parameters	**Value**
p : Population	300
m : Number of selected sites	30
e : Number of elites sites	10
nep : Number of bees around elite sites	20
nsp : Number of bees around other selected points	10
Probability of Chemotaxis	0.25
Max of time to be in Neutral Medium	3,6,9

Table 4.3 Comparison of computational times between the basic Bees Algorithm and the Bacterial Bees Algorithm: Instance 1-25

Instance	Optimum	The basic Bees Algorithm's running time (sec)	The Bacterial Bees Algorithm's running time (sec)	Δ_{avg}
1	913	84.93	5.64	-93.35
2	1225	93.27	8.48	-90.90
3	537	372.86	34.26	-90.81
4	2094	79.22	8.76	-88.95
5	990	18.84	2.37	-87.44
6	6955	162.12	11.94	-92.64
7	6324	183.48	14.78	-91.94
8	6865	89.84	8.61	-90.41
9	16225	218.92	13.48	-93.84
10	9737	236.08	28.61	-87.88
11	17465	268.28	19.66	-92.67
12	19312	420.22	22.10	-94.74
13	29256	836.41	415.14	-50.37
14	14377	363.51	100.32	-72.40
15	26914	744.05	32.84	-95.59
16	72317	2793.78	99.75	-96.43
17	78623	1463.90	164.46	-88.77
18	74310	4218.73	184.20	-95.63
19	77122	2683.74	189.84	-92.93
20	63229	3164.42	413.02	-86.95
21	77774	8265.32	501.64	-93.93
22	100484	5058.46	821.04	-83.77
23	135618	5091.31	513.36	-89.92
24	119947	12187.37	920.15	-92.45
25	128747	8018.63	421.20	-94.75

Table 4.4 Comparison of computational times between the basic Bees Algorithm and the Bacterial Bees Algorithm: Instance 26-50

Instance	Optimum	The basic Bees Algorithm's running time (sec)	The Bacterial Bees Algorithm's running time (sec)	Δ_{avg}
26	108	26.23	1.51	-94.25
27	64	49.87	1.76	-96.48
28	15	9.70	0.56	-94.23
29	47	28.42	0.85	-97.00
30	98	21.81	1.99	-90.89
31	6575	16.35	16.35	-94.82
32	4098	6.46	6.46	-96.59
33	5468	20.52	20.52	-95.72
34	2648	24.71	24.71	-86.47
35	5290	13.45	13.45	-95.69
36	19732	91.98	91.98	-93.29
37	17349	35.92	35.92	-95.16
38	24499	49.21	49.21	-94.73
39	19008	160.75	160.75	-84.58
40	19611	56.36	56.36	-97.39
41	57640	186.06	186.06	-89.09
42	81462	703.61	703.61	-93.90
43	65134	352.30	352.30	-94.67
44	78139	349.72	349.72	-97.51
45	66579	1071.16	1071.16	-83.68
46	64451	715.55	715.55	-87.40
47	113999	840.03	840.03	-89.56
48	74323	1087.91	1087.91	-90.39
49	110295	642.71	642.71	-91.12
50	95616	614.53	614.53	-90.83

Table 4.5 Comparison of computational times between the basic Bees Algorithm and the Bacterial Bees Algorithm: Instance 51-75

Instance	Optimum	The basic Bees Algorithm's running time (sec)	The Bacterial Bees Algorithm's running time (sec)	Δ_{avg}
51	0	69.47	2.69	-96.12
52	0	9.58	0.34	-96.44
53	0	9.68	0.34	-96.53
54	0	7.23	0.30	-95.87
55	0	11.96	0.58	-95.11
56	2099	600.86	20.77	-96.54
57	2260	6608.07	42.33	-99.36
58	4936	879.75	64.06	-92.72
59	3784	391.77	32.81	-91.63
60	3289	315.44	38.12	-87.92
61	20281	6719.99	28.08	-99.58
62	13403	-	44.89	-
63	19771	8178.16	48.76	-99.40
64	24346	3606.48	38.65	-98.93
65	14905	768.62	30.01	-96.10
66	65386	9918.82	46.52	-99.53
67	65756	14290.57	78.06	-99.45
68	78451	7351.29	46.81	-99.36
69	81627	6685.56	55.16	-99.17
70	68242	5340.08	48.21	-99.10
71	90486	5524.07	32.52	-99.41
72	115249	7296.84	28.64	-99.61
73	68529	6413.35	32.04	-99.50
74	79006	10390.73	29.15	-99.72
75	98110	5371.25	29.24	-99.46

Table 4.6 Comparison of computational times between the basic Bees Algorithm and the Bacterial Bees Algorithm: Instance 76-100

Instance	Optimum	The basic Bees Algorithm's running time (sec)	The Bacterial Bees Algorithm's running time (sec)	Δ_{avg}
76	0	7.22	0.45	-93.71
77	0	9.76	0.38	-96.11
78	0	4.94	0.17	-96.52
79	0	14.34	0.41	-97.14
80	0	21.47	0.88	-95.90
81	684	1712.20	109.90	-93.58
82	172	242.46	12.21	-94.96
83	798	536.20	24.05	-95.51
84	617	759.82	50.51	-93.35
85	776	-	69.54	-
86	10262	2023.87	73.88	-96.35
87	18646	35266.67	71.24	-99.80
88	10021	4322.38	34.97	-99.19
89	25881	6875.27	68.23	-99.01
90	8159	4970.31	134.19	-97.30
91	47683	16140.77	67.32	-99.58
92	43004	8827.16	26.67	-99.70
93	55730	5041.33	47.46	-99.06
94	59494	5168.17	44.05	-99.15
95	42688	16425.03	51.44	-99.69
96	126048	7291.68	32.46	-99.55
97	114686	4035.00	28.56	-99.29
98	112102	6259.16	25.17	-99.60
99	98206	7351.92	34.27	-99.53
100	157296	12608.67	30.82	-99.76

Table 4.7 Comparison of computational times between the basic Bees Algorithm and the Bacterial Bees Algorithm: Instance 101-125

Instance	Optimum	The basic Bees Algorithm's running time (sec)	The Bacterial Bees Algorithm's running time (sec)	Δ_{avg}
101	0	2.52	0.39	-84.45
102	0	23.97	0.87	-96.38
103	0	52.76	3.06	-94.21
104	0	12.05	0.36	-97.02
105	0	14.48	0.64	-95.57
106	0	116.55	4.22	-96.38
107	516	2371.98	45.01	-98.10
108	3354	3960.22	42.14	-98.94
109	0	47.61	1.63	-96.57
110	0	78.43	2.95	-96.24
111	31478	6447.61	32.97	-99.49
112	21169	-	2715.94	-
113	27077	10165.80	47.69	-99.53
114	19648	5685.70	41.38	-99.27
115	13774	16498.10	77.32	-99.53
116	46770	19830.87	120.20	-99.39
117	50364	11686.70	50.40	-99.57
118	25460	6249.10	56.16	-99.10
119	66707	8363.85	27.74	-99.67
120	69019	11290.97	88.94	-99.21
121	122266	3994.61	45.92	-98.85
122	82456	14371.13	35.28	-99.75
123	75118	10906.37	78.39	-99.28
124	73041	7879.30	43.35	-99.45
125	104531	9016.24	31.20	-99.65

Table 4.8 The Bacterial BA's computational results for 50 job problem: Ins 1-50

Instance	Optimum	Time (sec)
1	2134	4.42
2	1996	37.17
3	2583	4.39
4	2691	4.21
5	1518	15.61
6	26276	25.70
7	11403	18.45
8	8499	24.69
9	9884	16.75
10	10655	12.55
11	43504	35.77
12	36378	108.26
13	45383	31.93
14	51785	100.15
15	38934	20.81
16	87902	45.27
17	84260	-
18	104795	110.84
19	89299	91.88
20	72316	81.63
21	214546	104.20
22	150800	71.04
23	224025	80.51
24	116015	75.30
25	240179	79.30

Instance	Optimum	Time (sec)
26	2	6.13
27	4	1.78
28	755	35.20
29	99	15.47
30	22	3.84
31	9934	59.99
32	7178	157.57
33	4674	28.25
34	4017	41.71
35	6459	105.46
36	34892	82.40
37	22739	48.44
38	29467	103.55
39	49352	70.42
40	26423	63.20
41	71111	100.31
42	90163	80.56
43	84126	136.60
44	123893	213.70
45	79883	99.61
46	157505	90.09
47	133289	64.11
48	191099	69.15
49	150279	84.44
50	198076	72.16

Table 4.9 The Bacterial BA's computational results for 50 jobs: Ins 51-100

Instance	Optimum	Time (sec)
51	0	2.03
52	0	1.05
53	0	0.80
54	0	1.76
55	0	1.11
56	1258	30.42
57	3679	166.67
58	2522	27.12
59	3770	112.66
60	5904	108.53
61	25212	110.53
62	17337	136.06
63	30729	801.66
64	18082	220.47
65	25028	-
66	76878	345.07
67	85413	247.89
68	92756	253.95
69	77930	237.34
70	74750	235.40
71	150580	120.25
72	131680	140.74
73	98494	248.18
74	135394	119.88
75	135677	202.20

Instance	Optimum	Time (sec)
76	0	2.21
77	0	2.42
78	0	0.74
79	0	1.39
80	0	2.62
81	816	168.94
82	4879	147.27
83	973	456.12
84	508	33.55
85	3780	-
86	20751	1117.18
87	36053	917.55
88	28268	286.95
89	28846	128.34
90	15451	619.72
91	89298	399.84
92	66340	228.07
93	61060	308.16
94	42453	342.25
95	56522	380.86
96	177909	71.45
97	139591	83.08
98	148906	148.30
99	179264	79.04
100	120108	85.52

Table 4.10 The Bacterial BA's computational results for 50 jobs: Ins 101-125

Instance	Optimum	Time (sec)
101	0	1.16
102	0	1.08
103	0	0.70
104	0	1.33
105	0	8.27
106	0	3.83
107	1717	1228.14
108	0	7.73
109	6185	-
110	1295	551.41
111	27310	141.93
112	15867	124.66
113	35106	239.44
114	15467	-
115	10574	196.75
116	35727	129.03
117	71922	308.46
118	65433	139.12
119	106043	195.25
120	101665	763.26
121	78315	178.43
122	119925	110.67
123	101157	193.80
124	139488	268.94
125	110392	113.13

Table 4.11 The Bacterial BA's's computational results for 100 jobs: Ins 1-50

Instance	Optimum	Time (sec)
1	5988	64.10
2	6170	157.76
3	4267	58.91
4	5011	50.56
5	5283	56.07
6	58258	257.42
7	50972	189.18
8	59434	232.80
9	40978	610.87
10	53208	658.07
11	181649	1952.42
12	234179	1751.46
13	178840	1229.41
14	157476	7311.54
15	172995	1553.44
16	407703	1812.34
17	332804	1915.74
18	544838	1675.21
19	477684	5713.15
20	406094	6641.12
21	898925	1295.57
22	556873	2016.39
23	539716	4604.94
24	744287	1710.34
25	585306	1271.63

Instance	Optimum	Time (sec)
26	8	31.33
27	718	618.15
28	27	58.47
29	480	536.29
30	50	31.66
31	24202	620.47
32	25469	816.09
33	32964	3853.48
34	22215	-
35	19114	1493.04
36	108293	1982.21
37	181850	3205.19
38	90440	1668.15
39	151701	3298.23
40	129728	1669.68
41	462324	4120.01
42	425875	21861.07
43	320537	8515.39
44	360193	-
45	306040	-
46	829828	2373.19
47	623356	2531.77
48	748988	2928.70
49	656693	1589.95
50	599269	3457.98

Table 4.12 The Bacterial BA's computational results for 100 jobs: Ins 51-100

Instance	Optimum	Time (sec)
51	0	3.73
52	0	3.60
53	0	3.36
54	0	5.28
55	0	3.33
56	9046	-
57	11539	553.83
58	16313	3281.77
59	7965	479.28
60	19912	1140.13
61	86793	-
62	87067	17957.30
63	96563	-
64	100788	-
65	56510	-
66	243872	-
67	401023	8698.46
68	399085	-
69	309232	-
70	222684	1182.16
71	640816	3431.82
72	611362	18813.50
73	623429	2445.46
74	584628	6492.95
75	575274	4910.05

Instance	Optimum	Time (sec)
76	0	5.75
77	0	2.49
78	0	5.11
79	0	4.39
80	0	4.15
81	1400	5896.30
82	317	927.67
83	1146	1334.17
84	136	42.30
85	284	452.16
86	66850	-
87	84229	-
88	55544	-
89	54612	4050.92
90	75061	-
91	248699	4867.05
92	311022	-
93	326258	-
94	273993	-
95	316870	10126.80
96	495516	9144.44
97	636903	6966.12
98	680082	8621.40
99	622464	5766.41
100	449545	63427.00

Table 4.13 The Bacterial BA's computational results for 100 jobs: Ins 101-125

Instance	Optimum	Time (sec)
101	0	3.84
102	0	3.05
103	0	3.73
104	0	5.29
105	0	3.15
106	0	19.09
107	1193	1914.77
108	0	97.45
109	232	428.21
110	0	302.39
111	159138	15012.82
112	174377	-
113	91171	-
114	168297	-
115	70190	-
116	370631	-
117	324437	-
118	246243	-
119	293576	6265.52
120	267326	-
121	471214	5411.93
122	570459	4136.22
123	397029	19223.10
124	431115	11192.80
125	560754	-

4.5 Summary

In this chapter, the Bees Algorithm is implemented to solve the single machine total weighted tardiness problem. A benchmark from the OR-LIBRARY is chosen to test its performance. The results show that the Bees Algorithm could successfully solve the 40 jobs benchmark. Also an enhanced Bees Algorithm called the Bacterial Bees Algorithm was proposed to improve the Bee Algorithm's performance. The computational results show that the enhanced algorithm could perform better than the basic one and some other well-known algorithms in the literature considered in this study.

CHAPTER 5

THE ADAPTIVE BEES ALGORITHM FOR WEIGHTED TARDINESS SCHEDULING WITH SEQUENCE-DEPENDENT SETUPS

5.1 Preliminaries

In this chapter, the Adaptive Bees Algorithm is proposed for solving machine total weighted tardiness with sequence-dependent setup times. Apparent Tardiness Cost with Setups (ATCS) heuristic is used to create a reasonably good starting solution together with a set of random solutions. The algorithm also adapts the idea of Neighbourhood change in Variable Neighbourhood Search (VNS), a meta-heuristic or framework for building heuristics.

The chapter is organised as follows: Section 5.2 describes single machine total weighted tardiness with sequence-dependent setup times and benchmark used in this study. Section 5.3 presents the Bees Algorithm enhanced to solve this problem. Its characteristics are described. In section 5.4, results are compared with the results derived from some existing research. The summary of this work is in Section 5.5.

5.2 The Weighted Tardiness Scheduling with Sequence-dependent Setups Problem

The objective of minimising the total weighted tardiness has been the subject of a very large amount of literature on scheduling, although sequence-dependent setups have not been so frequently considered. Setups usually correspond to preparing the production resources for the execution of the next job, and when the duration of such operations depends on the type of last completed job, the setups are called sequence-dependent. The presence of sequence-dependent setups greatly increases the problem difficulty since it prevents the application of dominance conditions used for simpler tardiness problems (Rubin and Ragatz 1995).

The Weighted Tardiness Scheduling with Sequence-dependent Setups problem corresponds to the scheduling of n independent jobs on a single machine. All jobs

are ready at time zero and released simultaneously. The machine is continuously available and can process only one job at a time. For each job j= 1,2,3,…,n, a processing time p_j, a due date d_j, and a weight w_j are given. A sequence-dependent setup time s_{ij} must be waited before starting the processing of job j if it is immediately sequenced after job i. The tardiness of a job j is defined as T_j = max $\{0, C_j - d_j\}$, where C_j is the completion time of job j. The objective of this scheduling problem is to find a processing order of all jobs that minimise the sum of the weighted tardiness $\sum_{i=1}^{n} w_i T_i$.

In 2003, Circirello (2003) has proposed a set of benchmark for the Weighted Tardiness Scheduling with Sequence-dependent Setups Problem. The version of the problem without setup time is NP-hard. The problem is further complicated by the fact that it takes variable amounts of time to setup the machine when switching between any two jobs. The completion time c_j of a job can be defined as :

$$c_j = \sum_{i \in Predecessors\ (j) \cup j} p_i + s_{Previous\ (i),i} \qquad \text{(Eq 5.1)}$$

where p_i, $s_{k,i}$ are the processing time of job i and the setup time of job i if it immediately follows job k, respectively. Predecessors(j) is the set of all jobs that come before job j in the sequence and previous(i) is the single job that immediately precedes job i. Three parameters characterising each problem

instance are the due date tightness factor τ, the due date range factor δ and the set up time severity factor η. The benchmark set is formed by the following parameter values: τ = {0.3, 0.6, 0.9}, δ = {0.25, 0.75} and η = {0.25, 0.75}. For each of the twelve combinations of parameter values, 10 problem instances with 60 jobs are generated. These 12 problem sets cover a spectrum from loosely to tightly constrained problem instances. The benchmark instances can be obtained at http://www.ozone.ri.cmu.edu/benchmarks.html

Recently, several approaches have been adopted to solve this benchmark dataset see, for examples, Simulated Annealing, Genetics Algorithms, and Tabu Search by Lin and Ying (2007), Ant Colony Optimisation Algorithm and Discrete Particle Swarm Optimisation Algotithm by Anghinolfi and Paolucci (2008), Discrete Differential Evolution Algorithm by Tasgetiren et al. (2009), Discret Electromagnetism-like Machanism by Chao and Liao (2012), General Variable neighbourhood serach by Kirlik and Oguz (2012), Scater Search by Guo and Tang (2011) and Exact Algorithm by Tanaka and Araki (2012).

5.3 The Adaptive Bees Algorithm

Apparent Tardiness Cost with Setups (ATCS) heuristic consists of two stages. The first stage is to estimate due date tightness, due date range, and setup time severity factors. These three factors define the problem instances and their respective makespan value. Next, two look-ahead parameter values (k_1 and k_2) are

calculated by using those three estimated values derived from first stage and then used to calculate a priority index, which determines the sequence of the jobs.

The due date tightness τ, due date range $\mathcal{R}$, and setup time severity factors η can be calculated as follows:

$$\tau = 1 - \frac{\bar{d}}{C_{max}} \quad \text{(Eq. 5.2)}$$

$$\mathcal{R} = \frac{d_{max} - d_{min}}{C_{max}} \quad \text{(Eq. 5.3)}$$

$$\eta = \frac{\bar{s}}{\bar{p}} \quad \text{(Eq. 5.4)}$$

$C_{\max}$ is the completion time after finishing processing last job added into the sequence, $\bar{d}$ is the average of the due dates, $d_{\max}$ and $d_{\min}$ represent the maximum

and the minimum of due dates, respectively, $\bar{s}$ denotes the average setup time and $\bar{p}$ denotes the average processing time.

Due to the sequence dependent setup times, the determining the maximum of the completion time beforehand is very difficult. An estimated $C_{\max}$ can be obtained by correlating the $C_{\max}$ value with the average processing time, the average setup time and a coefficient β:

$$C_{max} = n(\bar{p} + \beta\bar{s}) \qquad \text{(Eq. 5.5)}$$

Variability of setup times and the number of jobs in the instance would affect the value of β. By using the estimates of $\tau, \mathcal{R},$ and η, the parameters k_1 and k_2 can be calculated as follows:

$$k_1 = \begin{Bmatrix} 4.5 + R, & R \leq 0.5 \\ 6.0 - 2R, & R > 0.5 \end{Bmatrix} \qquad \text{(Eq. 5.6)}$$

$$k_2 = \frac{\tau}{2\sqrt{\eta}} \qquad \text{(Eq. 5.7)}$$

Finally, the priority index is determined with the following equation:

$$I_j(t,i) = \frac{w_j}{p_j} exp\left[-\frac{\max(d_j - p_j - t, 0)}{k_1 \bar{p}}\right] \exp\left[-\frac{s_{ij}}{k_2 \bar{s}}\right] \qquad \text{(Eq. 5.8)}$$

The above equation, *t* denotes, the current time, and *I* is the index of the job that is just processed. The ATCS rule separates the effect of the setup time. The priority of a job given by weighted shortest processing time ratio is exponentially discounted twice, once based on the slack and again based on the setup time. These two effects are scaled separately by the parameters k_1 and k_2, which jointly provide the look-ahead capabilities of the ATCS rule. The values of the parameters depend on the problem instance as they essentially perform the scaling (Lee at al. 1997; Kirlik and Oguz 2012).

According to Mladenovic and Hansen (1997), Hansen and Mladenovic (2001) and Hansen and Mladenovic (2003), Variable Neighbourhood Search (VNS) exploits systematically the following facts: A local minimum with respect to one neighbourhood structure is not necessarily so for another, a global minimum is a local minimum with respect to all possible neighbourhood structure, and for many

problems, local minima with respect to one or several neighbourhoods are relatively close to each other.

The last observation implies that a local optimum often provides some information about the global one. There might be several variables with the same value in both. However, it is usually not known which ones are such. A study of the neighbourhood of this local optimum is therefore in order, until a better one is found.

Reduced Variable Neighbourhood Search (RVNS) is a simple application of VNS. It is a pure stochastic search method. A set of neighbourhood structures $N_1(x)$, $N_2(x)$, …, $N_{kmax}(x)$ will be considered around the current point x. Usually, these neighbourhood structures will be nested. Then a point is chosen at random in the first neighbourhood. If its fitness value is lower than that of the incumbent, the search is recentered there. Otherwise, one proceeds to the next neighbourhood. After all neighbourhoods have been considered, one begins again with the first, until the stopping criteria is met. The description of the steps of the RVNS is as follows:

1) Find an initial solutions x and choose a stopping condition
2) Repeat the following until a stopping condition is met:

   2.1) k ⊆ 1

   2.2) Repeat the following steps until $k = k_{max}$

      Shake: take a solution randomly from N_k (x)

If this point is better than the incumbent, move there (x⊆ x'), and continue the search with N_1(k⊆ 1); otherwise, set k⊆ k+1

This study has used ATCS to generate a starting solution for the Adaptive Bees Algorithm. The Algorithm itself also generates a set of solutions randomly and adapts the idea of neighbourhood change within the search in VNS to find better solution and/or escape from local optima. During Neighbourhood search, the Bees Algorithm randomly generates the order of the neighbourhood search procedures. Six different procedures are used which are (See details in chapter 3 and 4):

1) Swapping between two jobs
2) Reversing job order
3) Swapping two groups of jobs
4) Swapping three job positions
5) Inserting first job to a new random position
6) Inserting last job to a new random position

After neighbourhood search, if the Bees Algorithm could find a better solution then it will apply the same neighbour hood procedure for the next iteration. Otherwise it will use the next procedure in the order. If the algorithm could not find a better solution in a certain times, it will abandon the site and create new potential solution randomly. The pseudo code of the Adaptive Bees Algorithm is given in Fig 5.1.

1. Initial population *(θ)* with *p-1* random solutions plus a solution by ATCS.

2. Evaluate fitness of the population.

3. While (stopping criterion has not been met).

4. Randomly create an order of neighbourhood procedures (k_n)

5. Select sites (*m*) for neighbourhood search.

6. Recruit bees for selected sites: elite sites (*e*) and other selected sites (*m-e*).

7. Evaluate the fittest values, if no improvement then changes the neighbourhood procedure to the next one in the order for next iteration. Otherwise perform the same procedure.

8. If no improvement for a certain time, save the best fitness and search for new potential solution; solution.

9. End while.

Figure 5.1 The pseudo code of the Adaptive Bees Algorithm

5.4 Experimental results

The Adaptive Bees Algorithm was implemented in Matlab. The 10 time experiment has been carried out on Dell laptop: Intel (R) Core (TM) 2 Duo CPU P8600 @2.40GHz 4 GB RAM and MacBook Pro: Intel Quad Core i7 2.3GHz. 6 GB RAM. Table 5.1 shows the parameters used for experiments for problem instance 1-40 and Table 5.2 shows the parameters used for experiments for problem instance 41-120. Table 5.3 - 5.8 show the results derived from the Adaptive Bees Algorithm (ABA), OBK which is the best-known solutions composed of the solutions generated by Simulated Annealing, Genetics Algorithms and Tabu Search by Lin and Ying (2007), ACO by Anghinolfi and Paolucci (2008), DPSO by Anghinolfi and Paolucci (2009), DDE by Tasgetiren et al. (2009), DEM by Chao and Liao (2012), GVNS by Kirlik and Oguz (2012), SS by Guo and Tang (2011), and EXACT by Tanaka and Araki (2012).

Performance of the algorithm was quantified by the average percentage of relation deviations which was computed as follows:

$$\Delta_{avg} = \sum_{i=1}^{10} \left(\frac{(ABA - OBK)}{OBK} \times 100 \right) / 10 \qquad \text{(Eq. 5.9)}$$

Table 5.9 - 5.14 show the average percentage of relation deviations of ABA, ACO, DPSO, DDE, DEM, GVNS, SS, and EXACT. The results show that the Adaptive Bees Algorithm was able to find 23 better solutions out of 120 instances than OBK and the same as DPSO, DDE, DEM, GVNS, and SS whereas ACO found only 22 better solutions. However, EXACT found 24 better solutions. There are 97 instances in total that the Adaptive Bees Algorithm could not performed better than OBK. Results of 94 out of those 97 instances were equal. Table 4.5 shows the average of the average percentage of relation deviations of all instances. It can be seen that the proposed algorithm could perform much better than ACO and DPSO and slightly better than GVNS. However, the EXACT perform better than other existing techniques including the Bees Algorithm.

Table 5.1 Parameters of the Adaptive Bees Algorithms to solve 1-40 instances

Parameters	**Value**
p : Population	400
m : Number of selected sites	50
e : Number of elites sites	10
nep : Number of bees around elite sites	30
nsp : Number of bees around other selected points	10

Table 5.2 Parameters of the Adaptive Bees Algorithms to solve 41-120 instances

Parameters	**Value**
p : Population	600
m : Number of selected sites	50
e : Number of elites sites	10
nep : Number of bees around elite sites	30
nsp : Number of bees around other selected points	15

Table 5.3 Comparison results of the Adaptive Bees Algorithm with best-known results from recent research: Ins 1-20

Instance	OBK	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
1	684	513	531	474	504	471	471	453	471
2	5082	5083	5088	4902	4902	4878	4854	4794	4878
3	1792	1769	1609	1465	1480	1430	1455	1390	1430
4	6526	6286	6146	5946	6026	6006	5906	5866	6006
5	4662	4263	4339	4084	4084	4114	4134	4054	4114
6	5788	7027	6832	6652	6712	6667	6667	6592	6667
7	3693	3598	3514	3350	3404	3330	3458	3267	3330
8	142	129	132	114	113	108	110	100	108
9	6349	6094	6153	5803	5894	5751	5778	5660	5751
10	2021	1931	1895	1799	1803	1789	1805	1740	1789
11	3867	3853	3649	3294	3078	2998	3190	2785	2998
12	0	0	0	0	0	0	0	0	0
13	5685	4597	4430	4194	4194	4068	4185	3904	4068
14	3045	2901	2749	2268	2375	2260	2340	2075	2260
15	1458	1245	1250	964	1030	935	953	724	935
16	4940	4482	4127	3876	3517	3381	3843	3285	3381
17	204	128	75	61	60	0	60	0	0
18	1610	1237	971	857	835	845	845	767	845
19	208	0	0	0	0	0	0	0	0
20	2967	2545	2675	2111	2167	2053	2058	1757	2053

Table 5.4 Comparison results of the Adaptive Bees Algorithm with best-known results from recent research: Ins 21-40

Instance	OBK	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
21	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0
24	1063	1047	1043	1033	1039	920	1044	761	920
25	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0
30	165	130	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0	0
37	755	400	186	107	116	46	296	46	46
38	0	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0

Table 5.5 Comparison results of the Adaptive Bees Algorithm with best-known results from recent research: Ins 41-60

Instance	OBK	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
41	71186	70253	69102	69242	69242	69242	69552	69102	69102
42	58199	57847	57487	57511	57511	57511	57511	57487	57487
43	147211	146697	145883	145310	145310	145310	145310	145310	145130
44	35648	35331	35331	35289	35289	35289	35289	35166	35289
45	59307	58935	59175	58935	58935	59025	58935	58935	59025
46	35320	35317	34805	34764	34764	34764	34887	34764	34764
47	73984	73787	73378	73005	73005	72853	73157	72853	72853
48	65164	65261	64612	64612	64612	64612	64688	64612	64612
49	79055	78424	77771	77641	77641	77833	77771	77641	77641
50	32797	31826	31810	31565	31565	31292	31519	31292	31292
51	52639	50770	49907	49927	49927	49761	50101	49761	49761
52	99200	95951	94175	94603	94603	93106	96225	93106	93106
53	91302	87317	86891	84841	84841	84841	87559	84841	84841
54	123558	120782	118809	119226	119226	119074	121228	118809	118809
55	69776	68843	68649	66006	66006	65400	66006	65400	65400
56	78960	76503	75490	75367	75367	74940	75079	74940	74940
57	67447	66534	64575	64552	64552	64575	64552	64552	64522
58	48081	47038	45680	45322	45322	45322	46324	45322	45322
59	55396	54037	52001	52207	52207	51649	53315	51649	51649
60	68851	62828	63342	60765	60765	61755	62783	60765	60765

Table 5.6 Comparison results of the Adaptive Bees Algorithm with best-known results from recent research: Ins 61-80

Instance	OBK	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
61	76396	75916	75916	75916	75916	75916	75916	75916	75916
62	44769	44869	44769	44769	44769	44769	44769	44769	44769
63	75317	75317	75317	75317	75317	75317	75317	75317	75317
64	92572	92572	92572	92572	92572	92572	92572	92572	92572
65	127912	126696	126696	126696	126696	126696	126696	126696	126696
66	59832	59685	59685	59685	59685	59685	59685	59685	59685
67	29390	29390	29390	29390	29390	29390	29390	29390	29390
68	22148	22120	22120	22120	22120	22120	22120	22120	22120
69	64632	71118	71118	71118	71118	71118	71118	64632	71118
70	75102	75102	75102	75102	75102	75102	75102	75102	75102
71	150709	145825	145771	145007	145264	145007	145290	145007	145007
72	46903	45810	43994	43904	43286	43286	44558	43286	43286
73	29408	28909	28785	28785	28785	28785	28785	28785	28785
74	33375	32406	30734	30313	29777	30136	30142	30136	30136
75	21863	22728	21602	21602	21602	21602	21758	21602	21602
76	55055	55296	53899	53555	53555	54024	55482	53555	53555
77	34732	32742	31937	32237	31817	31817	32931	31817	31817
78	21493	20520	19660	19462	19462	19462	20008	19462	19462
79	121118	117908	114999	114999	114999	114999	115644	114999	114999
80	20335	18826	18157	18157	18157	18157	18824	18157	18157

Table 5.7 Comparison results of the Adaptive Bees Algorithm with best-known results from recent research: Ins 81-100

Instance	OBK	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
81	384996	383485	383703	383485	383485	383485	383485	383485	383485
82	410979	409982	409544	409544	409479	409479	409479	409479	409479
83	460978	458879	458787	458752	458752	458752	458752	458752	458752
84	330384	329670	329670	329670	329670	329670	329670	329670	329670
85	555106	554766	555130	554993	554870	554766	554870	554766	554766
86	364381	361685	361417	361417	361417	361417	361837	361417	361417
87	399439	398670	398551	398670	398551	398551	398551	398551	398551
88	434948	434410	433519	433186	433186	433244	433244	433186	433186
89	410966	410102	410092	410092	410092	410092	410092	410092	410092
90	402233	401959	401653	401653	401653	401653	401653	401653	401653
91	344988	340030	343029	340508	339933	339933	340221	339933	339933
92	365129	361407	361152	361152	361152	361152	361250	361152	361152
93	410462	408560	406728	404548	403423	404917	405978	403423	404548
94	335550	333047	332983	333020	332941	332949	335106	332941	332983
95	521512	517170	521208	517011	516926	517646	519843	516926	517646
96	461484	461479	459321	457631	455448	457631	460140	455448	455488
97	413109	411291	410889	409263	407590	407590	413671	407590	407590
98	532519	526856	522630	523486	520582	520582	525439	520582	520582
99	370080	368415	365149	364442	363977	363977	369154	363518	363977
100	439944	436933	432714	431736	431736	432068	435064	431736	432068

Table 5.8 Comparison results of the Adaptive Bees Algorithm with best-known results from recent research: Ins 101-120

Instance	OBK	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
101	353408	352990	352990	352990	352990	352990	352990	352990	352990
102	493889	493936	493069	492748	492572	492572	493036	492572	492572
103	379913	378602	378602	378602	378602	378602	378602	378602	378602
104	358222	358033	357963	357963	357963	357963	358334	357963	357963
105	450808	450806	450806	450806	450806	450806	451249	450806	450806
106	455849	455093	455152	454379	454379	454379	455031	454379	454379
107	353371	353368	352867	352766	352766	352766	352766	352766	352766
108	462737	461452	460793	460793	460793	460793	461452	460793	460793
109	413205	413408	413004	413004	413004	413004	413408	413004	413004
110	419481	418769	418769	418769	418769	418769	418769	418769	418769
111	347233	346763	342752	342752	342752	342752	343953	342752	342752
112	373238	373140	369237	367110	367110	367110	372819	367110	367110
113	261239	260400	260176	260872	259649	259649	260077	259649	259649
114	470327	464734	464136	465503	464001	463474	463474	463474	463474
115	459194	457782	457874	457289	456904	457189	459538	456890	457089
116	527459	532840	532456	530803	530601	530601	533160	530601	530601
117	512286	506724	503199	502840	502840	503046	507474	502840	502840
118	352118	355922	350729	349749	349749	349749	353142	349749	349749
119	579462	573910	573046	573046	573046	573046	573541	573046	573046
120	398590	397520	396183	396183	396183	396183	398528	396183	396183

Table 5.9 Comparison of the average percentage of relation deviations: Ins 1-20

Instance	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
1	-25.00	-22.37	-30.70	-26.32	-31.14	-31.14	-33.77	-31.14
2	0.02	0.12	-3.54	-3.54	-4.01	-4.49	-5.67	-4.01
3	-1.28	-10.21	-18.25	-17.41	-20.20	-18.81	-22.43	-20.20
4	-3.68	-5.82	-8.89	-7.66	-7.97	-9.50	-10.11	-7.97
5	-8.56	-6.93	-12.40	-12.40	-11.75	-11.33	-13.04	-11.75
6	21.41	18.04	14.93	15.96	15.19	15.19	13.89	15.19
7	-2.57	-4.85	-9.29	-7.83	-9.83	-6.36	-11.54	-9.83
8	-9.15	-7.04	-19.72	-20.42	-23.94	-22.54	-29.58	-23.94
9	-4.02	-3.09	-8.60	-7.17	-9.42	-8.99	-10.85	-9.42
10	-4.45	-6.23	-10.98	-10.79	-11.48	-10.69	-13.90	-11.48
11	-0.36	-5.64	-14.82	-20.40	-22.47	-17.51	-27.98	-22.47
12	0	0	0	0	0	0	0	0
13	-19.14	-22.08	-26.23	-26.23	-28.44	-26.39	-31.33	-28.44
14	-4.73	-9.72	-25.52	-22.00	-25.78	-23.15	-31.86	-25.78
15	-14.61	-14.27	-33.88	-29.36	-35.87	-34.64	-50.34	-35.87
16	-9.27	-16.46	-21.54	-28.81	-31.56	-22.21	-33.50	-31.56
17	-37.25	-63.24	-70.10	-70.59	-100.00	-70.59	-100.00	-100.00
18	-23.17	-39.69	-46.77	-48.14	-47.52	-47.52	-52.36	-47.52
19	-100.00	-100.00	-100.00	-100.00	-100.00	-100.00	-100.00	-100.00
20	-14.22	-9.84	-28.85	-26.96	-30.81	-30.64	-40.78	-30.81

Table 5.10 Comparison of the average percentage of relation deviations: Ins 21-40

Instance	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
21	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
22	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
23	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
24	-1.51	-1.88	-2.82	-2.26	-13.45	-1.79	-28.41	-13.45
25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
26	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
27	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
28	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
29	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	-21.21	-100.00	-100.00	-100.00	-100.00	-100.00	-100.00	-100.00
31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
32	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
34	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
36	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
37	-47.02	-75.36	-85.83	-84.64	-93.91	-60.79	-93.91	-93.91
38	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
39	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 5.11 Comparison of the average percentage of relation deviations: Ins 41-60

Instance	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
41	-1.31	-2.93	-2.73	-2.73	-2.73	-2.30	-2.93	-2.93
42	-0.60	-1.22	-1.18	-1.18	-1.18	-1.18	-1.22	-1.22
43	-0.35	-0.90	-1.29	-1.29	-1.29	-1.29	-1.29	-1.41
44	-0.89	-0.89	-1.01	-1.01	-1.01	-1.01	-1.35	-1.01
45	-0.63	-0.22	-0.63	-0.63	-0.48	-0.63	-0.63	-0.48
46	-0.01	-1.46	-1.57	-1.57	-1.57	-1.23	-1.57	-1.57
47	-0.27	-0.82	-1.32	-1.32	-1.53	-1.12	-1.53	-1.53
48	0.15	-0.85	-0.85	-0.85	-0.85	-0.73	-0.85	-0.85
49	-0.80	-1.62	-1.79	-1.79	-1.55	-1.62	-1.79	-1.79
50	-2.96	-3.01	-3.76	-3.76	-4.59	-3.90	-4.59	-4.59
51	-3.55	-5.19	-5.15	-5.15	-5.47	-4.82	-5.47	-5.47
52	-3.28	-5.07	-4.63	-4.63	-6.14	-3.00	-6.14	-6.14
53	-4.36	-4.83	-7.08	-7.08	-7.08	-4.10	-7.08	-7.08
54	-2.25	-3.84	-3.51	-3.51	-3.63	-1.89	-3.84	-3.84
55	-1.34	-1.62	-5.40	-5.40	-6.27	-5.40	-6.27	-6.27
56	-3.11	-4.39	-4.55	-4.55	-5.09	-4.92	-5.09	-5.09
57	-1.35	-4.26	-4.29	-4.29	-4.26	-4.29	-4.29	-4.34
58	-2.17	-4.99	-5.74	-5.74	-5.74	-3.65	-5.74	-5.74
59	-2.45	-6.13	-5.76	-5.76	-6.76	-3.76	-6.76	-6.76
60	-8.75	-8.00	-11.74	-11.74	-10.31	-8.81	-11.74	-11.74

Table 5.12 Comparison of the average percentage of relation deviations: Ins 61-80

Instance	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
61	-0.63	-0.63	-0.63	-0.63	-0.63	-0.63	-0.63	-0.63
62	0.22	0.00	0.00	0.00	0.00	0.00	0.00	0.00
63	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
64	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
65	-0.95	-0.95	-0.95	-0.95	-0.95	-0.95	-0.95	-0.95
66	-0.25	-0.25	-0.25	-0.25	-0.25	-0.25	-0.25	-0.25
67	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
68	-0.13	-0.13	-0.13	-0.13	-0.13	-0.13	-0.13	-0.13
69	10.04	10.04	10.04	10.04	10.04	10.04	0.00	10.04
70	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-1.33
71	-3.24	-3.28	-3.78	-3.61	-3.78	-3.60	-3.78	-3.78
72	-2.33	-6.20	-6.39	-7.71	-7.71	-5.00	-7.71	-7.71
73	-1.70	-2.12	-2.12	-2.12	-2.12	-2.12	-2.12	-2.12
74	-2.90	-7.91	-9.17	-10.78	-9.70	-9.69	-9.70	-9.70
75	3.96	-1.19	-1.19	-1.19	-1.19	-0.48	-1.19	-1.19
76	0.44	-2.10	-2.72	-2.72	-1.87	0.78	-2.72	-2.72
77	-5.73	-8.05	-7.18	-8.39	-8.39	-5.19	-8.39	-8.39
78	-4.53	-8.53	-9.45	-9.45	-9.45	-6.91	-9.45	-9.45
79	-2.65	-5.05	-5.05	-5.05	-5.05	-4.52	-5.05	-5.05
80	-7.42	-10.71	-10.71	-10.71	-10.71	-7.43	-10.71	-10.71

Table 5.13 Comparison of the average percentage of relation deviations: Ins 81-100

Instance	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
81	-0.39	-0.34	-0.39	-0.39	-0.39	-0.39	-0.39	-0.39
82	-0.24	-0.35	-0.35	-0.36	-0.36	-0.36	-0.36	-0.36
83	-0.46	-0.48	-0.48	-0.48	-0.48	-0.48	-0.48	-0.48
84	-0.22	-0.22	-0.22	-0.22	-0.22	-0.22	-0.22	-0.22
85	-0.06	0.00	-0.02	-0.04	-0.06	-0.04	-0.06	-0.06
86	-0.74	-0.81	-0.81	-0.81	-0.81	-0.70	-0.81	-0.81
87	-0.19	-0.22	-0.19	-0.22	-0.22	-0.22	-0.22	-0.22
88	-0.12	-0.33	-0.41	-0.41	-0.39	-0.39	-0.41	-0.41
89	-0.21	-0.21	-0.21	-0.21	-0.21	-0.21	-0.21	-0.21
90	-0.07	-0.14	-0.14	-0.14	-0.14	-0.14	-0.14	-0.14
91	-1.44	-0.57	-1.30	-1.47	-1.47	-1.38	-1.47	-1.47
92	-1.02	-1.09	-1.09	-1.09	-1.09	-1.06	-1.09	-1.09
93	-0.46	-0.91	-1.44	-1.71	-1.35	-1.09	-1.71	-1.44
94	-0.75	-0.77	-0.75	-0.78	-0.78	-0.13	-0.78	-0.77
95	-0.83	-0.06	-0.86	-0.88	-0.74	-0.32	-0.88	-0.74
96	0.00	-0.47	-0.83	-1.31	-0.83	-0.29	-1.31	-1.30
97	-0.44	-0.54	-0.93	-1.34	-1.34	0.14	-1.34	-1.34
98	-1.06	-1.86	-1.70	-2.24	-2.24	-1.33	-2.24	-2.24
99	-0.45	-1.33	-1.52	-1.65	-1.65	-0.25	-1.77	-1.65
100	-0.68	-1.64	-1.87	-1.87	-1.79	-1.11	-1.87	-1.79

Table 5.14 Comparison of the average percentage of relation deviations: Ins 101-120

Instance	ACO_AP	DPSO	DDE	DEM	GVNS	SS	EXACT	ABA
101	-0.12	-0.12	-0.12	-0.12	-0.12	-0.12	-0.12	-0.12
102	0.01	-0.17	-0.23	-0.27	-0.27	-0.17	-0.27	-0.27
103	-0.35	-0.35	-0.35	-0.35	-0.35	-0.35	-0.35	-0.35
104	-0.05	-0.07	-0.07	-0.07	-0.07	0.03	-0.07	-0.07
105	0.00	0.00	0.00	0.00	0.00	0.10	0.00	0.00
106	-0.17	-0.15	-0.32	-0.32	-0.32	-0.18	-0.32	-0.32
107	0.00	-0.14	-0.17	-0.17	-0.17	-0.17	-0.17	-0.17
108	-0.28	-0.42	-0.42	-0.42	-0.42	-0.28	-0.42	-0.42
109	0.05	-0.05	-0.05	-0.05	-0.05	0.05	-0.05	-0.05
110	-0.17	-0.17	-0.17	-0.17	-0.17	-0.17	-0.17	-0.17
111	-0.14	-1.29	-1.29	-1.29	-1.29	-0.94	-1.29	-1.29
112	-0.03	-1.07	-1.64	-1.64	-1.64	-0.11	-1.64	-1.64
113	-0.32	-0.41	-0.14	-0.61	-0.61	-0.44	-0.61	-0.61
114	-1.19	-1.32	-1.03	-1.35	-1.46	-1.46	-1.46	-1.46
115	-0.31	-0.29	-0.41	-0.50	-0.44	0.07	-0.50	-0.46
116	1.02	0.95	0.63	0.60	0.60	1.08	0.60	0.60
117	-1.09	-1.77	-1.84	-1.84	-1.80	-0.94	-1.84	-1.84
118	1.08	-0.39	-0.67	-0.67	-0.67	0.29	-0.67	-0.67
119	-0.96	-1.11	-1.11	-1.11	-1.11	-1.02	-1.11	-1.11
120	-0.27	-0.60	-0.60	-0.60	-0.60	-0.02	-0.60	-0.60

Table 5.15 Comparison of the average of the average percentage of relation deviations of all instances

	ACO_AP	**DPSO**	**DDE**	**DEM**	**GVNS**	**SS**	**EXACT**	**ABA**
AVG	-3.34	-5.31	-6.78	-6.78	-7.51	-6.29	-8.33	-7.54

5.5 Summary

In this chapter, the Adaptive Bees Algorithm is presented. The proposed algorithm deploys ATCS heuristic and random technique to find a set of starting solutions and adapts the idea of neighbourhood change in VNS for the use of neighbourhood search procedure. It was applied to a Weighted Tardiness Scheduling with Sequence-dependent Setups problem. The results were compared to those obtained by Simulated Annealing (SA), Genetic Algorithms (GAs), Tabu Search (TS), Ant Colony Optimisation (ACO), Discrete Particle Swarm Optimisation (DPSO), Discrete Differential Evolution (DDE), Discrete Electromagnetism-like Mechanism (DEM), General Variable Neighbourhood Search (GVNS), Scatter Search (SS), and EXACT Algorithm. The results show that the proposed algorithm performs better than or as well as the others. However, EXACT performs better than the Adaptive Bees Algorithm.

CHAPTER 6

CONCLUSION

6.1 Contributions

The overall aim of this research was to develop the Bees Algorithm for single-machine scheduling and to improve the performance of the algorithm. The research contributions include:

- Different versions of the Bees Algorithm for single-machine scheduling.
- Enhancements to the basic algorithm, with proofs to show that the enhanced version is both more robust and efficient than the original.
- A number of neighbourhood search procedures to help the algorithm find better solutions faster.
- A new method of selecting potential solutions for the next iteration. The method helps significantly to improve the speed of the algorithm.

- Proofs that, for some benchmark problems, using a tool to generate good starting solutions might help the algorithm find the optimum faster than when starting solutions are randomly produced.

6.2 Conclusion

In this thesis, the feasibility of utilising the Bees Algorithm to solve machine scheduling problems has been demonstrated. Enhanced algorithms have been presented which improve the current state of the art in this research area. The key conclusions for each topic investigated are given below.

- The algorithm was applied to three complex scheduling problems with specific modifications for each. The algorithm was first enhanced to solve the problem of machine scheduling with common due date. The results were compared to those by the original version, which was the first Bees Algorithm developed for combinatorial domains and to the results by other well-known algorithms. This work has shown that the modified algorithm performs better than other existing techniques.
- The Bees Algorithm deploying the Negative Selection technique inspired by the Immune System delivers the most promising solutions for the next iteration. This improvement overcomes the drawback of keeping the fittest solution from each selected patch after the algorithm performs neighbourhood search in a combinatorial domain. The results have proved the efficiency and robustness of the new algorithm.

- The second application of the Bees Algorithm was to solve the total weighted tardiness problem. Providing a variety of neighbourhood search procedures to the Bees Algorithm and assigning different ways to deploy them could significantly reduce computational time. It is also important to study the nature of each benchmark to ensure a good match with the parameters used in the algorithm. This study found that the enhanced Bees Algorithm performs faster when assigned a small number of parameters together with a proper technique to avoid being trapped at local optima.
- Lastly, the Bees Algorithm was used to solve the problem of minimising total weighted tardiness with sequence-dependent setups, which is the most complicated of the three benchmarks. The study demonstrates that the algorithm needs a tool to help generate a good starting solution as well as a technique to deploy a set of neighbourhood search procedures. The results have shown that although the algorithm performs much better than some existing algorithms it is only slightly better than other algorithms.

6.3 Future work

Possible extensions that can be made to the work presented in this thesis include:

- Developing a tool that can generate a more uniform spread of starting solutions.
- Developing new local search algorithms for combinatorial domains.
- Using more complex models to improve the performance of the Bees Algorithm.
- Developing techniques to reduce the Bees Algorithm's computational time.
- Applying the enhanced Bees Algorithm to solve flow shop and job shop

scheduling problems.

- Applying the enhanced Bees Algorithm to schedule jobs using real-world data.
- Applying the enhanced Bees Algorithm to different types of scheduling problems, for example, class room timetabling.
- Combining other ideas from other techniques such as Exact, Scatter Search and Discrete Electromagnetism-like Mechanism to the Bees Algorithm to improve its performance.

References

Aarts, E. and Lenstra, J. K. 1997. *Local search in combinatorial optimization*. England: John Wiley & Sons Ltd,

Ahmad, A.S. 2012. *A study of search neighbourhood in the bees algorithm*. PhD Thesis, Cardiff University.

Aickelin, U., et al. 2004. Investigating Artificial Immune Systems for Job Shop Rescheduling in Changing Environments. *Poster Proceeding of ACDM*, Engineer's House, UK

Aickelin, U. and Dasguta, D. 2005. *Search Methodologies: Introductory Tutorials in Optimisation and Decision Support Techniques*. USA: Springer.

Anghinolfi, D. and Paolucci, M. 2008. A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem. *International Journal of Operations Research 2008;5(1)*, pp44–60.

Anghinolfi, D. and Paolucci, M. 2009. A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *Operational Research* 193, pp. 73 – 85.

Baker K.,R., and Scudder G.,D. 1989a. On the assignment of optimal due dates. *Operational Research* 40, pp. 93-95.

Baker K.,R., and Scudder G.,D. 1989b. Sequencing with earliness and tardiness penalties: a review. *Operational Research* 38, pp. 22-36.

Besten M.,D., et al. 2000. Ant Colony Optimization for the total weighted tardiness problem. *In Parallel Problem Solving from Nature PPSN VI* 1917, pp. 611-620.

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*, pp. 25-39.

Biskup, D. and Cheng, T.C.E. (1999). Multiple-machine scheduling with earliness, tardiness and completion time penalties. *Computers and Operations Research*, 26, pp. 45–57.

Biskup D., and Feldmann M. 2001. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers & Operations Research* 28, pp. 787-801.

Bonabeau E., et al. 1999. *Swarm Intelligence: from Natural to Artificial Systems*. New York: Oxford University Press.

Cagnina, L. 2004. Particle swarm optimization for sequencing problems: a study case. *Proceeding of the 2004 Congress on Evolutionary Computation*. USA, pp. 536-541

Camazine, S. and Sneyd, J. 1991. A model of collective nectar source selection by honey bees: self-organization through simple rules. *Theoretical Biology* 149, pp.547-571.

Camazine, S. et al. 1999. House-hunting by honey bee swarms: collective decisions and individual behaviours. *Insects soc*. 46:348-360.

Camazine, S. et al. 2003. *Self-Organization in Biological Systems*, Princeton: Princeton University Press.

Cerny, V. 1985. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications 45***,** pp. 41–51.

Chandrasekaran, M. et al. 2006. Solving Job Shop Scheduling Problems using Artificial Immune System. *Int J Adv Manuf Techno 1*, pp. 580-593.

Chao, C.W. and Liao, C.J. 2012. A discrete electromagnetism-like mechanism for single machine total weighted tardiness problem with sequence-dependent setup times. *Applied Soft Computing* 12, pp. 3079-3087.

Chen, M.H., et al. 2012. The design of Self-evolving Artificial Immune System II for Permutation Flow-shop Problem. *World Academy of Science, Engineering and Technology* 65.

Cheng, T. C. E. and Kahlbacher, H. G. 1991. A proof for the longest-job-first policy in one-machine scheduling. *Naval Research Logistics*, 38, pp. 715 – 720.

Cicirello, V.A. 2003. *Weighted tardiness scheduling with sequence-dependent setups a benchmark library*. Technical Report of Intelligent Coordination and Logistics Laboratory Robotics Institute, Carnegie Mellon University, USA

Clarke, G.M. and Cooke, D. 2004. *A basic course in statistics*. 1929-Chichester. Wiley.

Coello, C.A. et al. 2003. Use of an Artificial Immune System for Job Shop Scheduling. *Artificial Immune Systems, Lecture Notes in Computer Science 2787*, Springer, pp. 1-10.

Congram, R.K, et al. 1998. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing 14*, 2002, pp. 52-67.

Crauwels, H. et al. 1998. Local search heuristics for the single machine total weighted tardiness scheduling problem, *INFORMS Journal on Computing 10*, pp.341-350.

Croes, G.A. 1958. A method for solving traveling salesman problems. *Operations Res. 6*, pp., 791-812.

de Jong, K. A. 2006. *Evolutionary computation: a unified approach*. Cambridge: MIT Press.

de Castro L. N. and Von Zuben, F. J. 2001. *aiNet : An Artificial Immune Network for data analysis*. In Data Mining : A Heuristic Approach, USA: Idea Group Publishing.

de Castro L. N. and Von Zuben, F. J. 2002. Learning and Optimisation Using the Clonal Selection Principle. *IEEE Transactions on Evolutionary Computation, Special Issue on Artificial Immune Systems* (IEEE) **6** (3). pp. 239–251.

de Castro, L. N., and Timmis, J. 2002. *Artificial Immune Systems: A new Computational Intelligence Approach.* Springer.

Dorigo, M. et al. 1999. Ant algorithms for discrete optimization. *Artificial Life*, 5, 2, pp. 137-172.

Dorigo, M., et al. 1991. *Positive feedback as a search strategy*. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy.

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

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

Feldmann, M., and Biskup, D. 2003. Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. *Computers & Industrial Engineering*, 44, No. 2, pp. 307-323.

Ferrolho, A., and Crisostomo, M. 2006. Genetic algorithm for the single machine total weighted tardiness problem. In *2006 1ST IEEE International Conference on E-Learning in Industrial Electronics*, pp. 17-22.

Forrest S., et al. 1994. Self-nonself discrimination in a computer. *IEEE Symposium on Research in Security and Privacy*, pp. 202–212.

Garey, M. R. and Johnson, D. S. 1979. *Computers and Intractability: A Guide to Theory of NP-Completeness*, San Francisco: Freeman.

Ghanbarzadeh, A. 2007. *THE BEES ALGORITHM A Novel Optimisation Tool.* PhD Thesis, Cardiff University.

Glover, F. and Laguna, M. 1997. Tabu Search. Massachusetts:Kluwer Academic Publishers.

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

Guo, Q. and Tang, L. 2011. A new scater search approach for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. *Fourth International Workshop on Advanced Computational Intelligence*, Wuhan, Hubei, China, pp. 19-21.

Hall N.G., et al. 1991. Earliness/tardiness scheduling problem. Deviation of completion times about a restrictive common due date. *Operation Research* 39. pp. 847-856.

Hansen, P., and Mladenovie, N. 2001. Variable neighborhood search: principles and applications. *Operational Research* 30, pp 449-467.

Hansen P., and Mladenovie, N., 2003. *A Tutorial on Variable Neighborhood Search*. Les Cahiers du GERAD, HEC Montreal and GERAD.

Hart, E. and Ross, P. 1999. An Immune System Approach to Scheduling in Changing Environments. *Proc. Of the Genetic and Evolutionary Computation Conference, pp. 1559-1566*.

Hino, C. M. et al. 2005. Minimizing earliness and tardiness penalties in a single-machine problem with a common due date. *European Journal of Operational Research* 160, pp. 190-201.

Hofmeyr, S.A. and Forrest, S. 2000. Architecture for an Artificial Immune System. *Evolutionary Computation* 7, pp. 45-68.

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

Hoogeveen, J., A., and Van de Velde, S.L. 1991. Scheduling around small common due date. *Operational Research* 55, pp. 237-242.

Huang, C.,L., and Tung. 2006. Using mutation to improve discrete particle swarm

optimization for single machine total weighted tardiness problem. In *World Automation Congress (WAC2006)*. pp.1-6.

Jerne, N.K. 1974.Towards a network theory of the immune system. *Ann. Immunol*. (Inst. Pasteur), 125C, pp. 373-389.

Kanet, J. J. 1981. Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly* 28, pp. 643–651.

Karaboga, D. 2005. *An idea based on honey bee swarm for numerical optimization*, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey.

Karaboga, D., and Basturk, B. 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, *Journal of Global Optimization* 39, pp. 459–471.

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

Kellegoz, T., et al. 2008. Comparing efficiencies of genetic crossover operations for one machine total weighted tardiness problem. *Applied Mathematics and Computation* 199, pp. 590 – 598.

Kennedy, J. and Eberhart, R. 1995. Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks (ICNN'95),* Vol. IV, pp.1942- 1948.

Kennedy, J. and Eberhart, R. 1997. A discrete binary version of the particle swarm optimization algorithm. *Proc. of the 1997 conference on Systems, Man, and Cybernetics (SMC'97)*, pp.4104-4109.

Kim, J. and Bentley, P. 1999. The Artificial Immune Model for Network Intrusion Detection. *7th European Congress on Intelligent Techniques and Soft Computing*.

Kim, D.,H., et al. 2007. A hybrid genetic algorithm and bacterial foraging approach for global optimization. *Information Sciences* 177, pp. 3918-3937.

Kirlik, G., and Oguz, C. 2012. A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine. *Computers & Operations Research* 39(7), pp. 1506-1520.

Kirkpatrick, S., et al. 1983. Optimization by Simulated Annealing. *Science* 220 (4598), pp. 671–680.

Koc, E. 2010. The Bees Algorithm Theory Improvement and Application. PhD Thesis, Cardiff University.

Lee, Y.,C., and Zomaya, A.,Y. 2007. *Immune system support for scheduling*. Advances in Applied Self-Organizing Systems, M. Prokopenko (ed), pp. 247-270 (Chapter 11), Springer, London.

Lee, Y.H., et al. 1997. A heuristic to minimise the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, pp. 45-52.

Lenstra, J.K. et al. 1977. *Complexity of machine scheduling*. Ann of Discret math.

Li, J.Q., et al. 2013. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. *Applied Mathematical Modelling.*

Lin, S.W., and Ying, K.C. 2007. Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. *International Journal of Advanced Manufacturing Technology*, pp. 1183-1190.

Lu, T. 2012. A Danger Theory Based Mobile Virus Detection Model and Its Application in Inhibiting Virus. Journal of Networks 7, pp.1227-1232.

Matzinger, P. 1994. *Tolerance, danger and the extended family*. Annual Review Immunology, pp. 991–1045.

Matzinger, P. 2002. The danger model: a renewed sense of self. *Science*, pp. 301-305.

Mladenovi, N., and Hansen, P. 1997. "Variable neighborhood search". *Computers and Operations Research* **24** (11), pp1097–1100.

Nearchou, A. C. 2004. Solving the single machine total weighted tardiness scheduling problem using a hybrid simulated annealing algorithm. In *2004 2nd IEEE International Conference on Industrial Informatics, 2004. INDIN 2004*, pp. 513-516.

Nearchou, A. C. 2008. A differential evolution approach for the common due date early/tardy job scheduling problem, *Computers & Operations Research* 35(4), pp. 1329-1343.

Nearchou, A. C. 2006. An efficient meta-heuristic for the single machine common due date scheduling problem. *In 2nd I Proms Virtual International Conference.*

Passino, K.,M. 2002. Biomimicry of bacterial foraging for distributed optimization and control. *Control Systems IEEE* 22(3), pp. 52-67.

Pan, Q. K., et al. 2006. A Discrete Particle Swarm Optimization Algorithm for Single Machine Total Earliness and Tardiness Problem with a Common Due Date. In *IEEE Congress on Evolutionary Computation 2006*, Vancouver, BC, Canada, pp. 3281-3288.

Pan, Q. K., et al. 2010. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Information science 181(12), pp.2455-2468.

Pinedo, M. 2008. Theory, Algorithms, and Systems, Springer, New York,

Pham, D. T., et al. 2005. The Bees Algorithm, Technical Note, Manufacturing Engineering Centre, Cardiff University, UK.

Pham, D. T., et al. 2006a. The Bees Algorithm, a novel tool for complex optimisation problems. In *Proc. 2nd Int. virtual conf. On intelligent production machines and systems (IPROMS)*. Oxford: Elseiver UK.

Pham, D.T., et al. 2006b. Optimising neural networks for identification of wood defects using the Bees Algorithm. *Proceedings of the IEEE International Conference on Industrial Informatics*, Singapore, pp. 1346-1351.

Pham, D.T., et al., 2006c. Optimisation of the weights of multi-layered perceptrons using the bees algorithm", *Proceedings of 5th Int. Symposium on Intelligent Manufacturing Systems*, Sakarya, Turkey, pp. 38-46.

Pham, D.T., et al. 2006d. "Some Applications of the Bees Algorithm in Engineering Design and Manufacture", *In: Proc. of the IEEE Int. Conf. on Industrial Informatics*, Singapore, pp. 62-70.

Pham, D.T., and Ghanbarzadeh, A. 2007. Multi-Objective Optimisation using the Bees Algorithm, *Proc. of the 3rd Virtual Int. Conf. on Innovative Production Machines and Systems (IPROMS 2007)*, Whittles (Dunbeath, UK) and CRC Press (Boca Raton, FL), pp 529-533.

Pham, D.T., et al 2007a. Data Clustering Using the Bees Algorithm, *Proc 40th CIRP International Manufacturing Systems Seminar*, Liverpool, UK.

Pham D.T., et al. 2007b, Using the Bees Algorithm to Schedule Jobs for a Machine, Proceedings 8th international Conference on Laser Metrology, *CMM and Machine Tool Performance (LAMDAMAP)*. Cardiff, UK, Euspen, pp. 430-439.

Pham, D.T., et al. 2012. An Application of the Bees Algorithm to the Single-Machine Total Weighted Tardiness Problem, *Proc of Eighth International Symposium on Intelligent and Manufacturing Systems*, Autonomous Service and Manufacturing Ststems, IMS 2012, September, 27-28, Antalya, Turkey

Robert, Y. and Vivien, F. 2010. Introduction to scheduling. CRC press, USA.

Rubin P.A, and Ragatz GL. 1995. Scheduling in a sequence dependent setup environment with genetic search. Computers & Operations Research 1995;22(1). pp.85–99.

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 am home: how the scouts in a honey bee swarm perceive the completion of their group decision making. *Behav Ecol Sociobiol 54*, pp. 511-520.

Smith W.E. 1956. Various optimizers for single-stage production. Naval Research Logistics Quarterly. v3. pp. 59-66.

Storn, R, and Price, K. 1997. Differential Evolution-a simple and efficient heuristic for global optimisation over continuouse spaces. Journal Global Optimisation. Number 4, pp. 341-359.

Talebi, J., et al. 2009. An efficient scatter search algorithm for minimizing earliness and tardiness penalties in a single machine scheduling problem with common due date. In *IEEE Congress on Evolutionary Computation, 2009. CEC '09*, pp. 1012-1018.

Thammano, A., and Phu-ang, A. 2013. A hybrid Artificial Bee Colony Algorithm with local search for flexible job-shop scheduling problem. Procedia Computer Science 20. pp. 96-101.

Tanaka, S. and Fujikuma, S. 2008. An efficient exact algorithm for general single-machine scheduling with machine idle time. *4th IEEE Conference on Automation Science and Engineering* (IEEE CASE 2008), pp 371–376.

Tanaka, S., et al. 2009. An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling* 12, pp. 575– 593.

Tanaka, S., and Fujikuma, S. 2012. A dynamic-programming-based exact algorithm for single-machine scheduling with machine idle time. *Journal of Scheduling* 15. pp. 347–361.

Tanaka, S. 2012. *An exact algorithm for the single-machine earliness-tardiness scheduling problem*. Just-in-Time Systems, Springer, pp 21-40.

Tanaka, S., and Sato, S. 2013. An exact algorithm for the precedence-constrained single-machine scheduling problem. *European Journal of Operational Research* 229, pp. 347–361.

Tanaka, S., and Araki, M. 2013. An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times. *Computers & Operations Research*, volume 40, pp. 344-352.

Tasgetiren, M.,F., et al. 2004. Particle swarm optimization algorithm for single machine total weighted tardiness problem, In *Congress on Evolutionary Computation CEC2004*, volume 2, pp.1412-1419.

Tasgetiren, M.,F., et al. 2009. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. *Computers & Operational Research*, volume 36, pp. 900- 915.

Varela, F.J., and Coutinho, A. 1991. Second Generation Immune Networks. *Imm. Today* 12, pp. 159-166.

Von Frisch, K. 1967. *Bees: Their Vision, Chemical Senses and Language*, (Revised ed.) Cornell University Press, N.Y., Ithaca.

Watanabe, Y. and Ishiguro, A. 1999. Decentralised Behaviour Arbitration Mechanism for Autonomous Mobile Robots Using Immune Network. In *Artificial Immune System and Their Applications*. Springer, pp.187-209.

Webster, S., et al. 1998. A genetic algorithm for scheduling job families on a single machine with arbitrary earliness/tardiness penalties and an unrestrictive common due date. *Production Research*, volume 39, pp. 2543-2551.

Zhang, F. and QI, D. 2012. A Positive Selection Algorithm for Classification. *Journal of Computational of Computational Information Systems*, pp. 207-215.

Zhang, R., et al. 2013. A Danger-Theory-Based Immune Network Optimization Algorithm. *The Scientific World Journal 13*.

Zhong, Y., et al. 2011. Hybrid artificial bee colony algorithm with chemotaxis behavior of bacterial foraging optimization problem. In *Seventh international Conference on Natural Computation (ICNC)*, pp. 1171–1174.