AN INVESTIGATION ON THE ANT-BASED HYPER-HEURISTIC FOR CAPACITATED VEHICLE ROUTING PROBLEM AND TRAVELING SALESMAN PROBLEM

Thesis submitted to the University of Nottingham for the degree of Doctor of Philosophy

by

Zalilah Abd Aziz

School of Computer Science and Information Technology

The University of Nottingham

March 2013

Table of Contents

Chapter 1 Introduction

Chapter 2 Literature Review

Chapter 3 The Vehicle Routing Problem and the Traveling Salesman Problem

Appendix

References

List of Figures

List of Tables

Abstract

A brief observation on recent research of routing problems shows that most of the methods used to tackle the problems are using heuristics and metaheuristics; and they often use problem specific knowledge to build or improve solutions. In the last few years, research on hyper-heuristic has been investigated which aims to raise the generality of optimisation systems. This thesis is concerned with the investigation of ant-based hyper-heuristic. Ant algorithms have been applied to vehicle routing problems and have produced competitive results. Therefore, it is assumed that there is a reasonable possibility that ant-based hyper-heuristic could perform well for the problem.

The thesis first surveys the literature for some common solution methodologies for optimisation problems and explores in some detail the ant algorithms and ant algorithm hyper-heuristic methods. Furthermore, the literature specifically concerns with routing problems; the capacitated routing problem (CVRP) and the travelling salesman problem (TSP). The thesis studies the ant system algorithm and further proposes the ant algorithm hyper-heuristic, which introduces a new pheromone update rule in order to improve its performance. The proposed approach, called the ant-based hyper-heuristic is tested to two routing problems; the CVRP and TSP. Although it does not produce any best known results, the experimental results have shown that it is competitive with other methods. Most importantly, it demonstrates how simple and easy to implement low level heuristics, with no extensive parameter tuning. Further analysis shows that the approach possesses learning mechanism when compared to random hyper-heuristic. The approach investigates the number of low level heuristics appropriate and found out that the more low level heuristics used, the better solution is

generated. In addition an ACO hyper-heuristic which has two categories of pheromone updates is developed. However, ant-based hyper-heuristic performs better and this is inconsistent with the performance of ACO algorithm in the literature.

In TSP, we utilise two different categories of low level heuristics, the TSP heuristics and the CVRP heuristics that were previously used for the CVRP. From the observation, it can be seen that by using any heuristics for the same class of problems, ant-based hyper-heuristic is seen to be able to produce competitive results. This has demonstrated that the ant-based hyper-heuristic is a reusable method. One major advantage of this work is the usage of the same parameter for all problem instances with simple moves and swap procedures. It is hoped that in the future, results obtained will be better than current results by using better intelligent low level heuristics.

Acknowledgement

First and foremost, I offer my sincerest gratitude to my supervisor, Professor Dr Graham Kendall, who has supported me with his patience and knowledge in completing my whole thesis and without him this thesis, too, would not have been completed or written. One simply could not wish for a better or friendlier supervisor.

I wish to express my acknowledgement to Kementerian Pengajian Tinggi and University of Technology Mara for the financial sponsorship throughout this study. In my daily work, I have been blessed with friendly and cheerful fellow friends in ASAP group who have made my time spent at the University of in Nottingham an enjoyable experience. I offer my regards and blessings to all of those who have supported me in many ways during the completion of this thesis.

Lastly, I wish to express my heartfelt thanks to my family, my parents, husband, Baharudin and my six lovely children, Iman, Qutub, Habeebah, Zahraa, Khadijah and Hasif for their endless patience, love and encouragement. I am glad that they had an enjoyable and great time in England. Many thanks and may God bless us.

Declaration

I hereby declare that this thesis has not been submitted, either in the same or different form, to this or any other university for a degree.

Signature:

Chapter 1

Introduction

1.0 Background and Motivation

Optimisation problems consist of a large set of problems and these problems are defined in a class of problems called combinatorial optimisation problems. Lawler (1976) defined combinatorial optimisation problems as follows:

"*Combinatorial optimisation is the mathematical study of the arrangement, grouping, ordering, or selection of discrete objects, usually finite in numbers.*"

The simplest approach to solve combinatorial optimization problems is to perform an exhaustive search to all possible solutions, and return the best. However, for some problems, the number of possible solutions is too many for an exhaustive search to be a practical option. For example a 10 city travelling salesman problem has about 181,000 possible solutions and a 20 city problem, has about 10,000,000,000,000,000 possible solutions (Michalewicz and Fogel 2004). This type of combinatorial explosion is what limits the option to use an exhaustive search as the size of the problem instance increases. To measure the efficiency of search methods, the amount of computing resources needed to perform the procedure which include the computing space and

time are taken into account. A search method is considered efficient (tractable) if it can perform for any given number of inputs in a reasonable time (polynomial time algorithm). However, if it can perform for small sizes of input and for larger sizes of input, the running time becomes impractical; it is considered inefficient (intractable) or known as exponential time algorithm. Many of combinatorial optimization problems are considered *NP*-hard which often cannot be solved to optimality within polynomial time. More information on *NP* complete is presented in chapter 2.

Among combinatorial problems which have been heavily studied are routing problems and probably among the famous elements of these routing problems are the vehicle routing problem (VRP) and travelling salesman problem (TSP). Researchers and practitioners have developed models and algorithm that give them the ability to solve these problems. In general, routing problems involved in optimizing a route of courier for a set of vehicles or traveling salesman go round some customers or cities and returning back to a point of departure. The Vehicle Routing Problem (VRP) was introduced by Dantzig and Ramser in 1959. It is a combinatorial problem (Cordeau and Laporte 2005). The VRP is the generalization of the TSP and therefore, considered to be in the class of *NP*-hard problems (Pardalos et al. 2002, Braysy et al. 2004). The objective of VRP is to search for lowest cost routes from a depot to sets of other cities or customers. It has been largely researched because of its importance in logistic and supply chains management. The TSP is a problem of finding the shortest possible tour to visit each city exactly once and real-world examples such as manufacturing, telecommunications, logistics and many more. The history of TSP was believed to be found in 1920 in Vienna (Applegate et al. 1998). It is known to be in the class *NP*-hard problems (Karp 1972).

A brief observation of the recent research on routing problems shows that most of the methods used to tackle the problems are using heuristics and metaheuristics. These methods comprise from simple heuristics to complicated metaheuristics and often use problem specific knowledge to build or improve solutions (Braysy and Gendreau 2005a; 2005b). Some of these methods are rarely implemented because they are complicated and not easy to code; too many parameters are used which are difficult to understand and not often reusable (Cordeau et al. 2002). This motivates the investigation of developing simple heuristics (such as simple swaps and moves) to solve these routing problems.

Heuristics methods do not guarantee optimal solutions, however they often obtain good or near optimal solutions at relatively low computational cost. An example of simple heuristic method is the hill climbing technique. A good introduction to heuristic techniques is provided in Michalewicz and Fogel (2004). One of the limitations in traditional heuristic methods (for example the simple descent method) is their inability to make progress after becoming trapped in local optimum (Hansen and Mladenovic 2003). Local optimum is the point search space where all solutions points in the neighborhood are worse than the current solution (Burke and Kendall 2005). To prevent this problem, more advance methods called metaheuristics have been researched. These methods have some form of learning mechanisms to store information as the search process progresses and can be combined with different concepts to explore the search space. Examples of these methods include greedy randomised adaptive search procedure (GRASP), variable neighbourhood search (VNS), simulated annealing, tabu search, genetic algorithms, memetic algorithms and ant colony optimisation. Metaheuristics often use problem

specific knowledge to build or improve solutions. It often requires sets of parameters to be tuned. For example, tabu search requires an appropriate length of the tabu list, simulated annealing requires an appropriate cooling schedule, genetic algorithms require a population size and crossover probability.

Metaheuristics have been shown to work well on certain instance. However, for other instances, it does not perform well and often, it is expensive to adapt to new instances and problems. Furthermore, metaheuristics are often time-consuming and knowledge intensive processes that require a deep understanding of the problem domain. This motivates the investigation of developing an algorithm that can produce good quality solutions across different instances and problems and which do not require extensive parameter tuning. In the last few years, research on hyper-heuristic has been investigated (see chapter 2). It is specifically designed to raise the generality of optimisation systems in such a way that the technique can be reused and applied to other different problems. In contrast to metaheuristics that operate on search space of solutions, hyper-heuristic operates on search space of heuristics. The idea of hyper-heuristic is to provide a way to combine a few simple heuristics or to construct new heuristics from previous existing heuristics to search for good solutions (Burke 2010a). Among hyper-heuristics developed are monte carlo hyper-heuristic (Ayob and Kendall 2003), choice function hyper-heuristic (Cowling et al. 2000), simulated annealing hyper-heuristic (Bai and Kendall 2005; Bai et al. 2007; Downsland et al. 2007), tabu search hyper-heuristic (Kendall and Mohd Hussin 2004b; Burke et al. 2003a; Burke and Soubeiga 2003 and Burke et al. 2005), genetic algorithm hyper-heuristic (Cowling et al. 2002a, 2002b, 2002d; Han et al. 2002; Han and Kendall 2003b, Ochoa et al. 2009

and Terashima-Marin et al. 2006) and ant algorithms hyper-heuristic (Burke et al. 2005b and Chen et al. 2007).

Ant algorithms are among the most recent metaheuristics developed (Dorigo and Di Caro 1999a). Ant-based hyper-heuristic are inspired by these algorithms. It is based on observation about ant behaviour; the foraging behaviour of how they are able to find the shortest path between food sources and their nest. Ant algorithms have been applied to vehicle routing problems and have produced competitive results (Bullnheimer 1999a, 1999c). Although ant-based hyper-heuristic has been applied to two different scheduling problems (Burke et al. 2005b, Chen et al. 2007), it has never been applied to routing problems. Therefore, it is assumed that there is a reasonable possibility that ant-based hyper-heuristic could perform well for the problem. This thus motivates the investigation of a more general approach based on ant algorithms that can solve different routing problems across different instances without extensive parameter tuning and by using simple to implement low level heuristics.

In ant system (Dorigo and Di Caro 1999a) and the previous ant algorithms hyper-heuristics (Burke et al. 2005b, Chen et al. 2007), the key properties of these approaches are pheromone and heuristics information (visibility) updating activities. In the algorithms, as ants travel, they deposit a chemical substance called pheromone. The amount of pheromone corresponds to the quality of the solution found by the ants; and visibility information represents some forms of heuristic information, which is combined with the pheromone value in order to decide which city to visit next (in this case, the TSP is used). This motivates the investigation of the influence of these properties in the solutions for routing problems.

Following the ant algorithms, ant colony optimisation (ACO) algorithms have been developed. The idea behind this approach is obtained from Dorigo and Di Caro (1999b; 1999c). ACO hyper-heuristic applies the same methodology as ant algorithms however it differs in the updating of the pheromone trail procedure. There are two procedures involved; the local and global update. The global update will use the best solution found at the current iteration to update the pheromone trail and the local update is performed after each ant performs a tour. This has lead to better improvements in the solutions obtained in several problems (Dorigo and Di Caro 1999b; 1999c). The varieties of updating rules in pheromone trail have motivated the investigation of these activities for improving the solutions for routing problems.

1.1 Scope and Aim of the Thesis

This thesis investigates the ant-based hyper-heuristic methodology. To observe the generality, we apply the proposed methodology to two routing problems; the capacitated vehicle routing problem (CVRP) and the travelling salesman problem (TSP). To our best knowledge, no other work in this area has been published in the literature. It is not our aim to produce the best known solutions, as this is not the overall aim of a hyper-heuristic approach. Rather, we aim to investigate how the ant-based hyper-heuristic is able to operate across different problems using simple to implement low level heuristics to produce competitive results when compared to other approaches. Most importantly, we will use the same parameter settings across the two problems we address in this thesis.

In order to accomplish these aims; our objectives are as follows:

- To place our work in the context of previous work by discussing the scientific literature of related combinatorial optimization problems and the methodologies that have been employed. These aims are discussed in chapter 2 along with an investigation on approaches to solve two routing problems (capacitated routing problem (CVRP) and travelling salesman problem (TSP). These aims are discussed in chapter 3.
- To introduce simple heuristics to solve routing problems. A brief observation of the recent research on routing problems shows that to solve these problems involve complicated heuristics and metaheuristics. It often uses problem specific knowledge to build or improve solutions which is rarely implemented because they are complicated and not easy to code; too many parameters are used which are difficult to understand and not often reusable. This research investigates whether simple heuristics (knowledge poor heuristics with simple swaps and moves) when combined together are able to generate good quality solutions. Furthermore, whether they are reusable in solving problems of same class. These heuristics are addressed in chapter 5 and chapter 6.
- To establish a hyper-heuristic algorithm based on ant algorithms that can produce good quality solutions across different instances and problems and which do not require extensive parameter tuning. This involves the observations on the functions and key properties of the previous ant system algorithms and ant

algorithms hyper-heuristics. Research on how the framework of these approaches is observed to form a basis for the general approach in solving the routing problems. The details are discussed in chapter 4.

- To introduce a new pheromone and visibility update rule in order to enhance the performance of ant based hyper-heuristic. The research investigates whether the new pheromone and visibility update rule can generate solutions that are good across all instances and different problems. The details are discussed in chapter 4.
- To develop an ACO hyper-heuristic to serve as mean of comparisons for the ant-based hyper-heuristic. It differs from the ant based hyper-heuristic in the updating of pheromone values. The research issue is to investigate to effect of local and global updating procedure. The details are discussed in chapter 4.
- To implement our approach on the capacitated vehicle routing problem (CVRP). The research issue is to see whether this approach can generate good solutions across different instances with simple to implement low level heuristics. This aim is addressed in chapter 5.
- To implement our approach on the travelling salesman problem (TSP). This research investigates whether our approach is able to solve another problem of a same class. This thus tested the generality of our approach and this aim is explored in chapter 6.

1.2 Outline of the Thesis

The thesis is presented as seven chapters. This chapter discusses the background and aims of the thesis. Chapter 2 outlines the main concepts and background for the work presented in this thesis. We discuss combinatorial optimization problems, solution methods for these problems; heuristics, metaheuristics and hyper-heuristic. Chapter 3 outlines discussion from domain perspectives; we discuss the capacitated vehicle routing problem and travelling salesman problem and related work for these problems.

We discuss the general design issue for the ant-based hyper-heuristic and ant colony (ACO) hyper-heuristic methodology in Chapter 4. We analyse the ant system algorithms to observe the functions and key properties of the algorithm. We further describe the similarities and differences between previous ant algorithm hyper-heuristic and our hyper-heuristic methodology in order to meet the objectives of this thesis.

In Chapter 5, we apply the proposed approach to the capacitated vehicle routing problem (CVRP). We carry out sets of experiments to determine appropriate parameter values. We test our approach on the capacitated vehicle routing problem and compare its performance with other approaches. In addition, we carry out experiments utilizing different sets of low level heuristics to determine a suitable number of low level heuristics. We develop a random hyper-heuristic to serve as a means of comparison for our ant based hyper-heuristic. Finally, the ACO hyper-heuristic was applied to observe the effect of different pheromone updates strategy.

In Chapter 6, we apply our approach to the travelling salesman problem (TSP) and compare it with other approaches. We investigate the effectiveness of the high level selector by applying it to two different categories of low level heuristics (TSP heuristics and CVRP heuristics). Furthermore, the ACO hyper-heuristic was applied.

Finally, in Chapter 7, we discuss the overall performance of the ant-based hyper-heuristic and outline directions for future research.

Chapter 2

Literature Review

2.0 Introduction

The aim of this chapter is to present necessary background information for the research in this thesis. We survey the literature for some of the common solution methodologies for optimisation problems. We discuss the definition of optimisation problems, the algorithm complexity; further present the solution methods that have been used to solve the optimisation problems. Among the methods discussed are genetic algorithms, tabu search, simulated annealing, the greedy randomized adaptive search (GRASP) and variable neighborhood search (VNS). We also describe ant algorithms in some detail as this forms the basis for the research presented in this thesis. Different versions of ant algorithms are discussed; the ant system, the elitist ant, the rank-based ant, the max-min ant and the ant colony system.

Furthermore, we discuss the concept of hyper-heuristic and survey some work done on hyper-heuristic methods. We distinguish between the two categories of hyper-heuristic; the heuristic selection and the heuristic generation. We end this chapter by discussing in detail the seminal work of ant algorithm hyper-heuristic. This emphasises the motivation to develop a general approach based on ant algorithm that can solve different problems across different instances without extensive parameter tuning and by using simple to implement low level heuristics.

2.1 Optimisation Problem

Optimisation approach is a process of finding the best selection, arrangement or sequencing to achieve some evaluation function. In optimisation problem, the evaluation function represents the quality of a solution and the objective is either to minimise or maximise the evaluation function. Optimisation problems consist of a large set of problems and we define these problems in a class of problems called combinatorial optimisation problems. Examples of combinatorial optimisation problems are scheduling problem, routing problem, assignment problem, resource allocation, travelling salesman problem and etc. Blum and Roli (2003) defined combinatorial optimisation $P = (S, f)$ as :

- *A set of variables* $X = \{x_1, ..., x_n\}$;
- *Variable domains* $D_1, ..., D_n$;
- *Constraints among variables;*
- *An objective function f to be minimised or maximised where*

  $f: D_1 X.....X D_n \rightarrow \Re$;

The set of all possible feasible assignments is:

$S = \{s = \{(x_1, v_1), ... , (x_n, v_n)\} \mid v_i \in D_i,$ *s satisfies all the constraints*$\}$. *S is usually a search (or solution) space, as each element of the set can be seen as a candidate solution. To solve a combinatorial problem one has to find a solution* $s^* \in S$ *with minimum objective function value, that is* $f(s^*) \leq f(s)\ \forall \in S$, s^* *is called a globally optimal solution of (S,f) and the set* $S^* \subseteq S$ *is called the set of globally optimal solutions.*

As an example, in a facility location problem, a location is required for depots or service centres and to be chosen from a set of potential locations. To seek optimal solution for an optimisation problem, exact algorithm will be employed. Some examples of exact methods are linear programming, branch and bound dynamic programming and langrangian relaxation method. However, when processing, some exact algorithms are not bounded by polynomial time complexity, thus making these algorithms inefficient and it is often computationally expensive to find optimal solution in terms of time and storage. In such problem, heuristic or metaheuristic approaches are usually used to achieve satisfactory solutions within reasonable time required.

2.1.1 Algorithms Complexity

To solve combinatorial optimisation problems, exhaustive search method is often used. Exhaustive search works by enumerating all possible solutions and choose the best among these solutions. As defined by Michalewicz and Fogel (2004), efficiency of an algorithm is considered as the amount of computing resources needed to perform the algorithm, which includes the computing space and time. The computing space is concerned with the amount of memory required to execute the algorithm and the computing time indicates the number of steps required to execute the algorithm in order to obtain a result. In the theory of algorithm complexity, the efficiency of an algorithm is expressed as the time complexity.

The time complexity function is described as the maximum time needed for an algorithm to find a solution. Garey and Johnson (1979) defined the complexity of a problem as measured by time complexity of the most efficient algorithm for the problem. An algorithm is considered efficient

(tractable) if it is a polynomial time algorithm where it can perform for any given number of inputs in a reasonable time. An algorithm with exponential time is considered to be inefficient (intractable) because it can only execute algorithm with small size of input and for larger size of inputs. The running time, thus becomes impractical.

2.1.2 The Classes of *P*, *NP* and *NP* complete

A problem that has polynomial time complexity is said to be efficient. All efficient problems are considered to be under class *P* (polynomial), which indicates that there is deterministic algorithm that can solve this class of problems in polynomial time. However there are some problems for which the flexibility is unknown. For this class of problems, it is said to be *NP* (non deterministic polynomial) problems which indicate that there is a non deterministic algorithm that can solve this class of problems in polynomial time. Therefore, we can say that $P \subseteq NP$. For example, for the traveling salesman problem (TSP), we do not know if the problem is feasible, which means that there is no polynomial time algorithm that will always determine the shortest route. However, there is a class of problems that cannot be proven for any efficient algorithms exist to solve it and therefore, it cannot be said to belong to *NP* class. This class of problems is referred to as *NP*-hard problems which mean 'at least as hard as *NP* problems'. Another class of *NP* is *NP* completeness. This class consists of all problems in *NP* which indicate that there is a polynomial time algorithm for the problems, thus implying that all other problems in *NP* are polynomial time solvability and therefore $P = NP$. Cook (1971), has introduced the concept of *NP* completeness which proves that satisfiability problem can be solved in polynomial time if $P = NP$. Exact methods are used to produce optimal solutions. However, for problems with a large

solution space, it will become computational expensive and methods such as heuristic and metaheuristics are used to produce near-optimal solutions in reasonable time.

2.2 Heuristics

The word heuristic is derived from a Greek word which means to find or to discover. It is a search technique that tries to find good quality solutions at a reasonable computational cost. Reeves (2003) defined a heuristic as:

"a technique which seeks good (i.e. near optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is".

A good introduction to heuristic techniques is provided in Michalewicz and Fogel (2004). Examples of heuristic methods are the hill climbing technique, greedy algorithm and simple descent method. The heuristic method starts with an initial solution. A neighborhood solution is explored and will be chosen only if the neighborhood solution is better that the current solution. This process will iterate until a stopping criteria is met.

One of the limitations of heuristics is their inability to make progress after becoming trapped in a local optimum (Hansen and Mladenovic 2003). Local optimum is points in the search space where all the solutions point in the neighborhood are worse than the current solution (Burke and Kendall 2005). To prevent this problem, more advanced techniques called metaheuristics have been developed.

2.3 Metaheuristics

Metaheuristics are considered as intelligent search methodologies. All metaheuristics have some forms of learning mechanism to store information as the search progresses and can be combined with different concepts to explore the search space. Voss (2001) defined a metaheuristisc as:

"A *metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or simple local search, or just a construction method. The family of metaheuristics includes, but is not limited to, adaptive memory procedures, tabu search, ant systems, greedy randomised adaptive search, variable neighborhood search, evolutionary methods, genetic algorithm, scatter search, neural networks, simulated annealing, and their hybrids.*"

In Glover and Kochenberger (2003), metaheuristics are defined as:

"*Solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space*".

Osman and Kelly (1996) defined metaheuristics as:

"A metaheuristic is an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces using learning strategies to structure information in order to find efficiently near-optimal solutions"

Osman (2000) defined metaheuristics as:

"A metaheuristic may combine intelligently different concepts for exploring the search space and uses learning strategies to structure information".

Among the first classifications of metaheuristics are the constructive or local search based (Blum and Roli, 2003, Bai, 2005). In a constructive method, we start with an incomplete solution and gradually add to it until we have a complete and hopefully feasible solution. At each decision point, a choice is made as to how best to add to the current, partial solution which will hopefully lead to a good quality and feasible solution. An example of a constructive algorithm is the nearest neighbor algorithm, in which a problem is represented as a graph and the algorithm will randomly pick an initial city and iteratively add the closest city among the remaining cities.

The local search based or sometimes referred to as the neighborhood search is an iterative search procedure, which starts from an initial feasible solution and progressively attempts to improve it by applying a series of moves. At every iteration, the search moves to another (normally feasible) solution, which is hoped to be an improvement on the current solution, although we

could accept worse moves under certain conditions. The acceptance rule to accept another solution is either the best-improvement which chooses the neighbor solution with the largest improvement or the first-improvement which chooses the first improved solution found.

Another classification of metaheuristics is single population and population-based methods. Single population methods only maintain a single solution at each iteration. Examples of these methods include greedy randomised adaptive search procedure (GRASP), variable neighbourhood search (VNS), simulated annealing and tabu search. Population-based methods maintain a population of solutions. These methods include genetic algorithms, memetic algorithms and ant colony optimisation. Blum and Roli (2003) have summarized the fundamental properties of meta-heuristics. Among them are:

i. Meta-heuristics are strategies that guide the search process
ii. The goal of meta-heuristics is to efficiently explore the search space in order to find near-optimal solution
iii. It constitutes meta-heuristics algorithms range from local search procedures to complex learning processes
iv. Meta-heuristics methods are approximate and non-deterministic
v. Meta-heuristics may incorporate mechanisms to avoid getting trapped in confined areas of search space
vi. Meta-heuristics permit abstract level of description
vii. Meta-heuristics are not problem-specific but may use domain-specific knowledge in the form of heuristics that will be controlled by an upper level strategy

viii. More advanced meta-heuristics use search experience embodied in some form of memory to guide the search.

One important concept discussed in Blum and Roli (2003), is the diversification and intensification strategy. Diversification strategy refers to the exploration of the search space and intensification strategy refers to the exploitation of the accumulated search experience. More information on metaheuristics is reviewed in Blum and Roli (2003), Glover and Kochenberger (2003) and Gendreau and Potvin (2005b). In the following sections we briefly review some of the common metaheuristics.

2.3.1 Genetic Algorithms

Genetic algorithms are population based evolutionary algorithms that can be used to solve optimization problems. It was first proposed by Fraser (1957). These algorithms are inspired by Darwin's theory of evolution. The basic idea is to produce a population of individuals and through a number of generations; the population will evolve, and improve, by mimicking natural selection. Reeves (2003) and Sastry et al. (2005) provide firm introduction to this methodology. Genetic algorithms start by randomly generating an initial solution. It employs genetic operators such as *selection* and *mutation* to manipulate individual solutions (*chromosome*). The chromosome is usually fixed and each position is a gene and contains a value. This value could be a binary digit, or it could be a much more complex data structure.

A crossover operator replaces some of the genes in one parent corresponding to the genes of another. At each generation, the crossover operator creates one or more offspring from two

existing parents. The selection operator will select the best solution as parents in order to survive for the fittest to survive from one generation to the next generation. Most commonly used method for selection is the *roulette-wheel* method which uses probability distribution. Mutation operator allows exploration of the search space in order to overcome the problem of two parents having the same value of genes. Genetic algorithms have been successfully been applied to many combinatorial optimization problem (Holland 1992, Aickelin 1998, Reeves 2003, Ross et al. 1998).

2.3.2 Tabu Search

Tabu search was first proposed by Fred Glover in 1987. Glover and Laguna (1997) defined tabu search as:

> *"Tabu search is a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality."*

Tabu search are used to prevent cycling when moving away from local optima through non-improving moves (Gendreau and Potvin 2005a). It escapes from local optimum by implementing an exploration search strategy guided by information from the previous search history. Tabu search starts as a traditional local search. It will iteratively move from one solution to another by selecting the best neighbour solution at each iteration until a termination condition is met. To prevent cycling, a short-term memory which is known as a *tabu list* is used to store the most recently visited solutions (or attributes to the solutions). Any moves that will take the search back to a tabu position is forbidden. The search process is only allowed to visit solutions that are

not members of the *tabu* list. Once a solution is visited, it will be added to the *tabu* list and one of the solutions that were already stored in the *tabu* list will be removed out from the list.

Two important strategies in tabu search; *intensification strategy*, which involves exploring the neighborhood of current solution and *diversification strategy*, which allows the search process to explore unvisited region of the search space in order to search out promising areas. The length of the tabu list or *tabu tenure* determines the behavior of the search process. Smaller tabu tenure indicates only small areas of the search space will be explored (*intensification*). Intensification strategy involves exploring more thoroughly the search space that seems beneficial, in order for the best solutions in these areas to be found. Larger tabu tenure allows the search process to explore larger areas of the search space, thus allowing the *diversification* strategies to be implemented. Diversification strategy involves in exploring the unexplored areas of the search space. This action is performed when there are repetitions of solutions.

An *aspiration criterion* is introduced to allow a tabu list to be revoked. This is necessary if the tabu list contains beneficial moves. An example of a simple aspiration criteria is a tabu move is allowed when it produces a solution that is better than the current solution. Other criteria for aspiration criteria are discussed in Reeves (2003). Gendreau and Potvin (2005a) outlined that the key rule for a tabu to be revoked is when cycling does not occur. In order to produce good performance of tabu search, appropriate parameter tuning for the size of tabu list is needed. Tabu search has successfully been applied to several applications (Skorin-Kapov 1990; Gendreau et al. 1994; Chambers and Barnes 1996; Glover and Laguna 1997; Burke et al. 1999 and Gaspero and Schaerf 2007). Detail overview on tabu search can be obtained from Hertz et al. (1995) and Gaspero and Schaerf (2001).

2.3.3 Simulated Annealing

Simulated annealing was introduced by Kirkpatrick et al. in 1983. It originates from statistical mechanics (Metropolis algorithm) and is among the oldest metaheuristics approach that provides a strategy to avoid local optimum. The gist behind Metropolis algorithm is that it will proceed in small steps from one state to the next state and the temperature prevents the algorithm from getting stuck by permitting the uphill moves (assuming minimisation problem) (Kirkpatrick et al. 1983).

Simulated annealing concepts are based on thermal process to obtain low energy states of a solid in a heat bath. The process consists of two steps: the temperature of the heat bath is increased to a maximum value where the solid melts. Then the temperature is decreased so that the particles cool until it converges to a steady state (Aarts et al. 2005). With regard to the search process, this algorithm allows for two important strategies: setting the temperature T initially high will allow the exploration of the search space (*diversification*); the particles cool themselves until they converge into a steady state, thus leading the search to converge to a local minimum (*intensification*). Appropriate cooling schedule is crucial to balance between intensification and intensification strategies (Osman 1993). However, the cooling schedule and the initial temperature value depend on the problem instances since the structure of the search space varies from one instance to other instances.

Simulated annealing always accepts improving solutions and non-improving solutions will be accepted with a certain probability. The algorithm starts by heuristically or randomly generating an initial solution. A temperature T (a control parameter of the cooling schedule) is initialised. While a termination condition is not met, at each iteration, a candidate move is randomly selected. The move will be accepted if it leads to a solution with an improved objective function, (for example, in the case of minimisation, $f(s') < f(s)$). However, a non-improving move ($f(s') \geq f(s)$) will always be accepted depending on the deterioration of the objective function value. The move will be accepted based on a probability given by $exp^{(-\Delta/T)}$ (where Δ is $f(s') - f(s)$, that is the difference between the current and new solution) and T is the value of current temperature. The algorithm is typically terminated when the temperature reaches zero.

Simulated annealing has been applied to a large number of application areas, often with good quality results being achieved. However, it sometimes requires a large amount of computation time (Aarts et al. 2005). It has successfully been applied to several problems such as vehicle routing problem (VRP) (Osman 1993), quadratic assignment problem (QAP) (Connolly 1990) and travelling tournament problem (TTP) (Lim et al. 2006). More detail descriptions on simulated annealing are discussed in Henderson et al. (2003); Suman and Kumar (2006).Variants of simulated annealing include the threshold accepting algorithm and the great deluge algorithm.

2.3.4 Greedy Randomized Adaptive Search (GRASP)

Greedy randomized adaptive search (GRASP) was first introduced by Feo and Resende (1989). It is an iterative process that combines constructive heuristics and local search. It consists of two procedures: solution construction and solution improvement. At each iteration, a solution is

otherwise value k will be incremented with 1 and a different neighborhood will be examined. Detail implementation and applications of the algorithm can be seen in Hansen and Mladenovic (2001, 2003, 2005). VNS has been successfully applied to several applications such as nurse rostering problems (Burke et al. 2004) and examination timetabling (Abdullah and Burke, 2006).

2.3.6 Ant Algorithms

Ant Algorithms are discussed here in some detail as it is the fundamental to work carried out in this thesis. An ant algorithm is considered as constructive heuristic, where it builds the solution from an initially empty solution (Dorigo and Stutzle 2003a). While navigating to and from the food source, ants deposit a substance called pheromone. This trail permits communication among the ants so that other ants can navigate to the food source utilizing the experience of others in finding a shorter route.

2.3.6.1 Ant Behavior

The ant algorithm is based on an observation about ant behavior (Dorigo and Di Caro 1999a; 1999b; 1999c). They live in a colony and their behavior is directed more to the survival of the colony, rather than individual survival. An important behavior of the ants is the foraging behavior and how they are able to find the shortest path between food sources and their nest despite being almost blind.

2.3.6.2 The Double Bridge Experiment

Deneubourg et al. (1990) show that path selection to a food source by the Argentine ant is based on self-organization. In their experiment, a food source is separated from the nest by a bridge with two equally long branches. Initially, no pheromone exists on the branches. Therefore, they have the same probability to be selected by the ants. Ants deposit pheromone as they travel from nest to food source and vice versa. The principle of the algorithm is the shorter the path, the more pheromone is left on the path. Ants will follow the path with a pheromone trail and will tend to choose the path with higher amounts of pheromone.

They conducted an experiment to investigate the behavior of these ants towards the laying of pheromone trails. In the first experiment, the two branches had an equal length. The ants were left to walk freely between the nest and the food source. The percentage of ants that chose these branches was observed over time. Eventually, all the ants used the same branch. This was explained as follows; as there was no pheromone on the two branches when the experiment started, the ants may choose any of the branches with equal probability. However, with the assumption that the time scale is taken as consideration, the probability of ants choosing a branch at a certain time depends on the total number of ants that used the branch until the time.

Since ants deposit pheromone while walking, the larger number of ants on a branch results in a larger amount of pheromone on that branch. This larger amount of pheromone will motivate more ants to choose that branch. As a result, the ants converge into a single path. This process is categorized as positive feedback loop where the probability of an ant choosing a path increases

with the number of ants (Dorigo et al. 1991).The second experiment set one of the branches to be twice as long as the other. In this experiment, the ants converge to the shorter branch.

2.3.6.3 Pheromone Update

Ant algorithms are based on a positive feedback mechanism (Dorigo et al. 1991). Depositing pheromone allows good solutions to be retained for reinforcement. However, to avoid stagnation (and premature convergence), inferior solutions, not very good solutions should not be ignored.

Pheromone evaporation is done in order to avoid the unbounded accumulation of pheromone. This enables ants to slowly forget its past action and explore new search directions without being influenced by past decisions. Ant algorithm is an iterative procedure as the amount of pheromone is transferred from one iteration to the next. After an ant has built a solution, the quality of that solution is used to compute the amount of pheromone the ant should lay on a particular edge. For example in the Travelling Salesman Problem (TSP), the pheromone trail refers to the desirability of visiting city *j* after visiting city *i*. The amount of pheromone will be measured based on quality of solution generated. In an ant system (AS), each ant lays some amount of pheromone inversely proportional to the solution generated (Dorigo et.al. 1996). Two different ways of updating the pheromone trail are performed. The iteration-best updating is where the selected ant is the ant that does the shortest tour in the current iteration while the global-best updating is the ant that does the shortest tour since the beginning of the algorithm.

For a dynamic problem (routing problem) where the characteristics of the problem change unpredictably during the course of the algorithm, pheromone updating is performed when there

is shorter path involved. Pheromone will be laid on that particular path, thus influencing the decision of future ants (Di Caro and Dorigo 1997; 1998).

2.3.6.4 Heuristic Information

Heuristic information helps the ant's decision-making process by exploiting problem specific knowledge. For a static problem, the knowledge is computed once during the initialization phase and it remains the same during the run of the algorithm. For example, in TSP (Dorigo et al. 1991), the heuristic information is the inverse of the distance ($1/d_{ij}$) between city i and city j, and the distance do not change as the algorithm executes. This has some advantages as it is easy to compute and needs to be computed only once. It is also computationally efficient, as the values do not have to be updated at each iteration. For a dynamic problem, the heuristic information is changing throughout the execution time of the algorithm. It has to be continually computed each time ant makes a tour, thus increasing the computational time.

2.3.6.5 Number of Ants

The number of ants in the algorithm is usually set depending on the dataset. For example for the TSP (Dorigo et al. 1991) the number of ants is set to be the same number of cities exist in the network. A single ant is capable of searching for solution. However, in Dorigo and Gambardella (1997) and Dorigo and Stutzle (2003b), a colony of ants is proven to produce good results. Ants cooperate by exchanging information through pheromone trail. For example in the work (Dorigo and Gambardella 1997), a comparison between colonies of cooperating ants with a colony of

non-cooperative ants was done. CPU time was used to measure the performance. It was found that cooperating ants have a higher probability to find quicker optimal solutions.

A single ant is capable of searching for solutions. However, in Dorigo and Gambardella (1997) and Dorigo and Stutzle (2003b), a colony of ants is proven to produce good solutions. Cooperating of ants is more desired particularly for geographically distributed system (Dorigo and Stutzle 2003b). This is because the difference of path length contributes to the difference of pheromone trail and can be exploited well by the colony of ants.

2.3.6.6 Ant System

The original ant algorithm was named as *Ant System* (AS) (Dorigo et al. 1991; 1996). Three different version of AS were developed: ant-density, ant-quantity and ant-cycle. The difference between these algorithms is the way in which the pheromone values are updated. Ant-density and ant-quantity manipulates local information. The trail intensity is updated after every move, as opposed to waiting for a tour to be completed. More detailed explanation on how these algorithms work can be found in Dorigo et al. (1991) and Maniezzo and Colorni (1994). More descriptions for this algorithm are presented in Chapter 4.

2.3.6.7 Elitist Ant System

An improvement on AS, called elitist Ant System (EAS) was introduced in Dorigo et al. (1991). The edge belonging to the best tour is rewarded with more pheromone values compared to other

edges. This activity is to provide additional reinforcement learning to good edges. Details can be found in Dorigo et al. (1991).

2.3.6.8 Ant System Rank-Based

Ant System Rank-Based (AS_{rank}) was introduced by Bullnheimer et al. in 1999. Ants are sorted according to their tour lengths and the amount of pheromone that is deposited is carried out according to the rank of the ants (Bullnheimer et al. in 1999b).

2.3.6.9 *Max-Min* Ant System

Max-Min Ant System was introduced in Stutzle and Hoos (2000). In their algorithm, the values of the pheromone trails have an upper boundary value, τ_{max}, and a lower boundary, τ_{min}. The boundary is in the range of $\tau_{min} \leq \tau_{ij} \leq \tau_{max}$. More details can be seen in Stutzle and Hoos (2000).

2.3.6.10 Ant Colony System (ACS)

The Ant colony system (ACS) improves the AS via two mechanisms (Dorigo and Gambardella 1997; Dorigo and Di Caro 1999b). The first mechanism uses an elitist strategy to update pheromone trails (only the ant that produced the best solution is allowed to update pheromone trails). Secondly, the ant chooses the next city to move to by using the *pseudo-random proportional rule.* More work on this algorithm can be seen in Alexandrov and Kochetov (1999); Bell and McMullen (2004); Stutzle and Dorigo (1999) and Den Besten et al. (2000).

2.4 Hyper-heuristic

Metaheuristics are capable of producing good quality solutions. However, they often need to perform some adjustment of relevant parameters in order to be applied to new problem or different problem instances. Furthermore metaheuristics are often time-consuming and knowledge intensive process that requires a deep understanding of the problem domain. According to Burke et al. (2003b):

"*Many state-of-the-art metaheuristics developments are too problem-specific or knowledge-intensive to be implemented in cheap, easy-to-use computer systems*".

Hyper-heuristic are specifically designed to raise the generality of optimisation systems in such a way that the technique can be reused and applied to other different problems. A key motivation of this method is to design a system that could automate and simplify the tuning of heuristics in an optimisation problem. Burke et al. (2003b) defined the aim of hyper-heuristic as:

"*The aim is not to develop a method which would 'beat' existing algorithms for a given optimisation problem, but instead to develop a method which is capable of performing well-enough, soon-enough, cheap-enough across a range of problems and domains*".

The idea of hyper-heuristic is; it provides a way to combine few simple heuristics or to construct new heuristics from previous existing heuristics to search for good solutions. These simple heuristics can be any *k*-opt moves, simple local searches that are problem dependent or rules

used by problem experts to construct solutions. Whenever new problem domains are to be solved, the hyper-heuristic algorithm will only have to replace the set of these simple heuristics and the evaluation function (solution quality).

Metaheuristics operate on search space of solutions. However, hyper-heuristic operate on search space of heuristics. Problem owners normally would prefer simple, easy to implement heuristics which do not require a large amount of resources and expertise for the development. Therefore they should be cheaper to implement and easy to use and can produce good quality solutions. For example in Cowling et al. (2002a), it has been demonstrated that hyper-heuristic has produced a much better quality than manually scheduled problem of personnel scheduling in one of UK academic institutions in a shorter time period (three weeks).

Hyper-heuristic approaches have started as early as in 1960s (Fisher and Thompson 1961, 1963; Crowston et al. 1963) (as discussed in Burke et al. 2010). The term hyper-heuristic was first used in 1997 (Denzinger et al. 1997) (discussed in Burke et al. 2010) to introduce the concept of combining several artificial intelligence methods. In 2000, the hyper-heuristic term was formally used to solve the scheduling of sales summit (Cowling et al. 2000). These approaches however previously were not classified as hyper-heuristic; instead these approaches gave a root to hyper-heuristic method. For example, in scheduling open-shop schedule (Fang et al. 1993), genetic algorithm was used to search the space of heuristics for the problem. It produces some best known results on the benchmark problems. Hart and Ross apply genetic algorithm to search for heuristic combination to solve the dynamic job-shop scheduling problems (Hart and Ross 1998). The method produced very good results on 12 instances. Gratch et al. presented a paper in 1993 which develop a method (COMPOSER system) to control the satellite communication schedules

which comprise a number of earth orbiting satellites and three ground stations (Gratch et al. 1993; Gratch and Chien 1996). There is a learning system that searches a space of possible control strategies. Statistics are used to evaluate the performance of the expected problem. In the following section, we will discuss the hyper-heuristic classifications.

2.4.1 Hyper-heuristic Classifications

Basically hyper-heuristic are divided into two; the nature of search space and the types of feedback received during the execution of the algorithm (Burke et al. 20101a). These classifications determine the methodologies for hyper-heuristic to solve optimisation systems. Figure 2.3 illustrate the framework for these classifications.

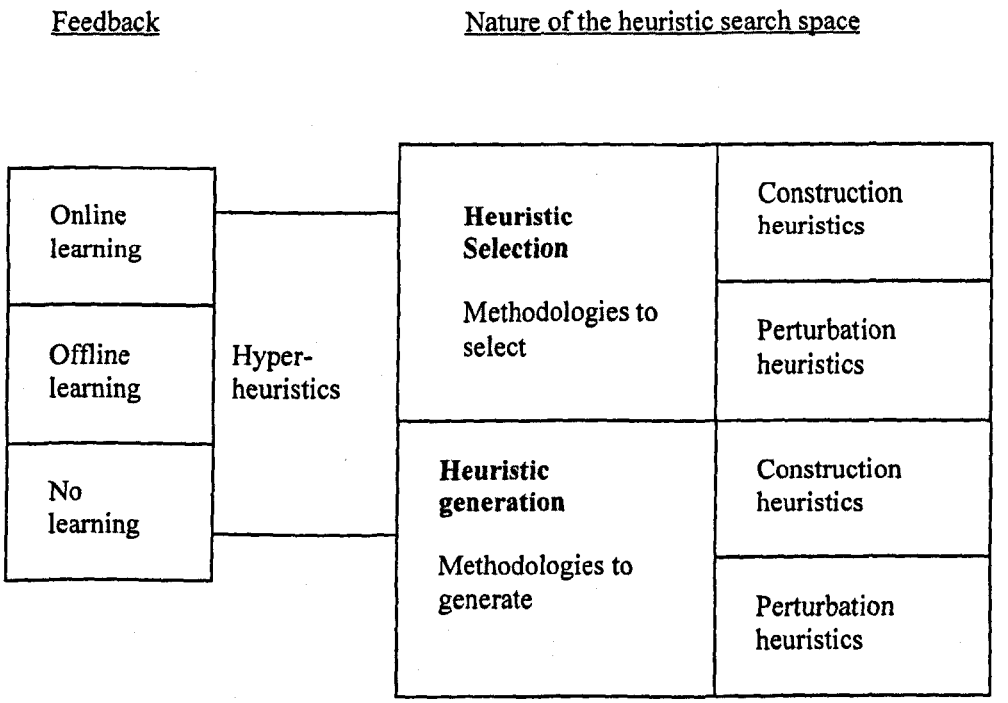

Figure 2.3: The framework for classification of the hyper-heuristic (source from: Burke et al. 2010b)

Hyper-heuristic is a learning algorithm when it involves some feedback during the search procedure. Feedback is important to determine the kind of hyper-heuristic method that we want to employ. For example in Cowling et al. (2001b) and Burke et al. (2003a), historical performance of each heuristics is considered to select the right heuristics at each decision point. There are three types of learning as follows:

- **Online Learning** – This type of learning requires the feedback to be recorded during the execution of an algorithm. An example of online learning is the use of metaheuristics or reinforcement learning as a high level selector to pick low level heuristics.
- **Offline Learning** - This type of learning requires the feedback to be recorded in the form of rules from training instances. This feedback is generally used to solve unseen instances.
- **Without Learning** – This type of learning does not require any feedback.

Hyper-heuristic operate on search spaces of heuristics instead of search space of solutions. A hyper-heuristic classification is considered in the section below.

2.4.1.1 Heuristic Selection

The first category is a method where heuristics choose heuristics; this method will select a good combination of low level heuristics to search for good solutions. There are several definitions for hyper-heuristic of this category in the literature. Among them are:

The non-domain data flow in the framework stores information about each low level heuristic. For example in the work of Kendall and Mohd Hussin (2004a), the non-domain information includes the heuristic number, recent changes in the evaluation function, CPU time taken by each heuristic to obtain a solution and the heuristics *tabu* status, indicating how long a heuristic should remain in the *tabu* list. For every iteration, at each decision point, a low level heuristic is selected. For example in Cowling, Kendall and Soubeiga (2000; 2002a; 2002b), the choice of the low level heuristic is based on the previous performance of each neighborhood. The performance of the low level heuristics is evaluated by calculating the weighted sum of three components; the recent performance of each low level heuristic, recent effectiveness of consecutive pairs of low level heuristics and the amount of time since the low level heuristic was last called. A balance of these three factors (represented as α, β, δ) allows the search space to be explored effectively.

Heuristic selection method was implemented based on constructive low level heuristics and perturbative low level heuristics. The first method selects low level heuristics to build solution incrementally from an empty solution until a complete solution is found. The second method selects low level heuristic to improve a current solution. Further discussion on these approaches will be discussed in the section below. The heuristic selection method was successfully tested on several optimization problems; production scheduling (Vaquez-Rodriquez et al. 2007), educational timetabling (Asmuni et al. 2004; Ross et al. 2004), bin packing problem (Bai et al. 2007), channel assignment problem (Kendall and Mohamad 2004a, 2004b), sales summit scheduling (Cowling et al. 2000), project presentation scheduling (Cowling et al. 2002a) and nurse rostering (Cowling et al. 2002a).

The basic framework of this method is to examine the available heuristics. These heuristics are usually human-created heuristics. They will then be used as base heuristics to generate another unknown heuristics. For example, we have a problem domain; the available heuristics will be observed to get the generalization of the heuristics. At this stage, the heuristics will be decomposed into their basic components. To generate new heuristics, the similar design issues as the genetic programming approach will be considered. Furthermore, we need to analyze how these heuristics will be applied to a dataset for a problem. In genetic programming, there exist a terminal set and function set. Terminal set consists of variables and constants of the programs; for example, commands such as forward, left or right. For the case of hyper-heuristic, this set of variables will describe the problem. They can be considered as input variables and changes their value as the problem state changes. Function set consists of the functions of program in genetic programming. Examples of function set can be the arithmetic operators or conditional operators. In hyper-heuristic framework, the function set will determine how the variables will be composed together. Fitness function will need to be identified for a particular problem. Finally, the genetic programming approach will be executed.

These categories of hyper-heuristic are based on constructive hyper-heuristic and perturbative heuristics. Both approaches select or generate different heuristics at different stages of the search procedure. The heuristics can be applied once or can be applied repetitively as long as it produces a better solution. Hyper-heuristic does not necessarily have to beat existing methods. Their prime motivation is to be a general method that works effectively across different problem domains.

2.4.2 Constructive Hyper-heuristic

Constructive hyper-heuristic refer to an approach that employs a set low level heuristics in order to produce high quality solutions from a given problem. They construct solutions from an empty solution by selecting low level heuristics at each decision point of solution construction. This process will continue until a complete solution is obtained. For example, in Qu and Burke (2005) and Burke et al. (2005a), to construct an educational timetabling, a list of constructive graph heuristics such as color degree, largest degree, largest enrollment, largest weight degree, saturation degree and random ordering method were used one by one to schedule slots into the timetable. First, the least penalty slots will be scheduled into the timetable. The next heuristic in the list will be used to schedule the remaining slots and the first set of the slots will be scheduled into the timetable. This procedure continues until a complete timetable is produced.

In Burke et al. (2007), a tabu search is used to search for good permutation of graph coloring heuristics within a graph-based hyper-heuristic to construct exam and course timetables. Asmuni et al. (2004) use fuzzy rules to select graph heuristics to construct the exam timetables. More examples of constructive hyperheuristic can be found in Terashima-Marin et al. (2006). Their approach to solve a large-scale university examination timetable problem is by using different sets of heuristic and a switch condition to move from one heuristic to another to construct a feasible timetable. Constructive hyper-heuristic has been applied to several domains of optimization problem such as vehicle routing problem (Garrido and Castro 2009), production scheduling (Vaquez-Rodriquez et al. 2007), educational timetabling (Asmuni et al. 2004, Ross et al. 2004) and bin packing problem (Bai et al. 2007).

2.4.3 Perturbative Hyper-heuristic

Perturbative hyper-heuristic refer to an approach that employs perturbative heuristics as low level strategies to solve a problem. They start from an initial solution and repeatedly select heuristics at each decision point in order to search for a better solution. Various work has been presented based on these perturbative hyperheuristic. In Kendall and Mohd Hussin (2004a), for example, an initial solution for the examination timetable is constructed using the largest degree or saturation degree. Tabu Search based hyper-heuristic will schedule the unscheduled exam using the low level heuristics, thus exploring the neighborhood to search for better solutions by selecting which heuristics to apply. Perturbative hyper-heuristic has been applied to several domains of optimization problem such as channel assignment problem (Kendall and Mohamad 2004a, 2004b), personnel scheduling (Cowling et al.2000), educational timetabling (Burke et al. 2003a, Burke et al. 2005a) and vehicle routing problem (Pisinger and Ropke 2007).

2.4.4 Low Level Heuristics

In a hyperheuristic framework, a set of low level heuristics is combined in order to be chosen by the hyper-heuristic. Kendall and Mohd Hussin (2004a) defines low level heuristics as:

"Low level heuristics are heuristics that allow movement through a solution space that require domain knowledge and are problem dependent."

These low level heuristics are typically simple, easy to implement and carry out one specific function although, of course, they can be as complex as the designer desires. Usually, they represent simple local search neighborhoods such as swap, move or rules that were used by the user to construct solutions for their problems. However, low level heuristics can also be complicated heuristics such as the metaheuristics. In a hyper-heuristic framework, these low level heuristics will generate a new solution together with other information such as changes in the evaluation function and CPU time taken to complete a task. For example, in the work of Bai (2005), 5 low level heuristics that were easy to implement and straightforward were used to solve the bin packing problem. By combining these heuristics, the search procedure was guided to promising direction.

2.4.5 Hyper-heuristic in the literature

In the following section, we will discuss the available hyper-heuristic approaches that are available in the literature.

2.4.5.1 Random Search Hyper-heuristic

The random search hyper-heuristic randomly selects the simplest and easiest heuristics to implement in hyper-heuristic approaches. Random search hyper-heuristic are usually used as benchmarks (comparison) for other hyper-heuristic. The drawback of these random search hyper-heuristic is that the performance of each heuristic cannot be measured since it is randomly generated.

There are four different approaches of random hyper-heuristic as defined in Cowling et al. (2000).

Simple random - this method will randomly pick a low level heuristic to apply at each iteration until a stopping condition is met. The pseudocode for this algorithm is described as in Figure 2.6:

Do
Select a low level heuristic at random and apply it once.
Until *stopping condition is met*

Figure 2.6: Simple random algorithm (source from: Cowling et al. 2000)

Random Descent – This method will equally pick a low level heuristic at random. It will be applied until there is no further improvement achieved (local optimum). This process repeats until a stopping condition is met. The pseudocode for this algorithm can be described as in Figure 2.7:

Do
Select a low level heuristic at random and apply it until no further improvement achieved.
Until *stopping condition is met*

Figure 2.7: Random descent algorithm (source from: Cowling et al. 2000)

Random Permutation – This method produces a random permutation of low level heuristics. The low level heuristics will then be applied repeatedly in the chosen permutation until a stopping condition is met. The pseudocode for this algorithm can be described as in Figure 2.8:

Produce a random permutation of low level heuristics available
Do
Apply the low level heuristic in the permutation sequence at once
Until *stopping condition is met*

Figure 2.8: Random permutation algorithm (source from: Cowling et al. 2000)

Random Permutation Descent – This method produces a random permutation of low level heuristics. The low level heuristics will then be applied in the chosen permutation until no further improvement is achieved (local optimum). The pseudocode for this algorithm can be as in Figure 2.9:

Produce a random permutation of low level heuristics available
Do
Apply the low level heuristic in the permutation sequence until no further improvement achieved.
Until *stopping condition is met*

Figure 2.9: Random permutation descent algorithm (source from: Cowling et al. 2000)

Random Descent (*RD*) hyper-heuristic has been applied in Kendall et al. (2002) to solve the real world problem of project presentations problem at the University of Nottingham. Two selection mechanisms were used: that is all moves (*AM*) and only improving moves (*OI*). However, all types of random hyper-heuristic as above were applied to a sales summit problem as in Cowling et al. (2001a) to investigate a more suitable strategy to automatically apply a parameter in a choice function hyper-heuristic.

2.4.5.2 Greedy Hyper-heuristic

Greedy hyper-heuristic consider the objective function produced by each low level heuristic. The solution will be compared with the current solution. The low level heuristic that produces the best solution will be applied as long as it produces an improvement.

For example, Ayob (2005) in her thesis applied three random approaches of greedy search hyper-heuristic. The random hyper-heuristics are:

- ***Random descent*** - this method randomly picks a low level heuristic and accepts the first improved heuristic.
- ***Random move*** - this method picks a randomly chosen heuristic without bothering about the quality of the solutions as long as it maintains the best solutions.
- ***Steepest descent*** – picks the best heuristic available in the neighborhood.

This type of hyper-heuristic is normally applied as a comparison for performance of algorithms.

2.4.5.3 Monte Carlo Hyper-heuristic

Monte Carlo hyper-heuristic was introduced in Ayob and Kendall (2003). It randomly picks a low level heuristic. Improved solutions are always accepted. In order to escape the local optimum, worse solutions are accepted with a certain probability based on Monte Carlo acceptance criteria. The probability decreases as the quality of the solution gets worse. A basic Monte Carlo hyper-heuristic algorithm from Ayob (2005) is described as in Figure 2.10:

Step 1: Initialisation

a. Choose a starting solution $S_0 \in S$;
b. Record the best obtained solution, $S_{best} = S_0$ and $f(S_{best}) = f(S_0)$;

Step 2: Choice and termination

a. *Choose $S_c \in n(S_0)$;*
b. *Compute $\delta = f(S_c) - f(S_0)$*
c. *If $\delta \leq 0$ then accept S_c and proceed to Step 3*
d. *Else: Accept S_c with a probability that decreases with increases in δ. If S_c is rejected and stopping condition = false, then return to step2(a);*
e. *Terminate by a stopping condition.*

Step 3: (Update)

Re-set $S_0 = S_c$ and if $f(S_c) < f(Sbest)$, return to Step1(b).
Return to Step2 if stopping condition= false.

Figure 2.10: Monte carlo hyper-heuristic (source from: Ayob 2005)

Ayob and Kendall (2003) investigate three types of acceptance criteria of the algorithm based on this approach: Linear Monte Carlo (LMC), Exponential probability function (EMC) and Exponential Monte Carlo with counter (EMCQ). They apply these hyper-heuristic to optimise the component placement sequencing in printed circuit board assembly. The objective is to improve the efficiency of the multi head placement machine. The Linear Monte Carlo acceptance probability is computed by $(M- \delta)$. M is a constant value between 0 and 100. The Exponential probability function is computed by $e^{-\delta}$.

The probability of accepting worse solution decreases as the δ decreased. New solution S_c will be accepted if the generated random number is less than $e^{-\delta}$. The Exponential Monte Carlo with counter (EMCQ) is computed by $e^{-\theta/\tau}$, where $\theta = \delta * t$ (t is a computation time) and $\tau = p(Q)$.

New solution S_c will be accepted if the generated random number is less than $e^{-\theta/\tau}$. For more detailed work on Monte Carlo hyper-heuristic, see Ayob (2005).

2.4.5.4 Choice Function Hyper-heuristic

Choice function hyper-heuristic was introduced in Cowling et al. (2000). The low level heuristics in a choice function hyper-heuristic are ranked based upon the historical performance of the heuristics, a learning mechanism that combines the intensification and diversification procedure during the search. A pseudocode for choice function hyperheuristic is shown in Figure 2.11:

Do
Select the low-level heuristic that maximizes choice function f and apply it.
Update choice function f's parameters using an adaptive procedure
Until *Stopping condition is met.*

Figure 2.11: Choice function hyper-heuristic (source from: Cowling et al. 2001a)

For example, in Cowling et al. (2001a), the choice function considers three different criteria: the recent performance of the low level heuristics (f_1), the recent improvement for consecutive pairs of the low level heuristics (f_2) and the amount of time elapsed since the given heuristic has been called (f_3). The value of f_1 and f_2 are considered as an element to guide the intensification procedure and f_3 provides an element of diversification, which guides the search into considering the low level heuristics that have not been applied for some time. Full detailed of this method is presented in Soubeiga (2003).

In terms of performance, Kendall et al. (2002) investigate the choice function hyper-heuristic and compared it with a range of heuristics and hyper-heuristic. They reported the choice function

hyper-heuristic can produce effective and realistic combination of low level heuristics. Further improvement on results produced by this method (choice function all moves) can be found in Cowling et al. (2002c). Choice function hyper-heuristic has been applied on several different problem domains such as the sales summit scheduling (Cowling et al. 2001a) and the nurse scheduling problem (Kendall et al. 2002)

2.4.5.5 Simulated Annealing Hyper-heuristic

Simulated annealing hyper-heuristic always accepts low level heuristic that produce improving solutions and low level heuristics with non-improving solutions will be accepted with certain probability. Initially, the probability of accepting new solution is high to allow more search space to be explored and it decreases as the search progresses. In Soubeiga (2003), the acceptance probability is controlled by a geometric cooling schedule. The pseudocode of a basic algorithm is as in Figure 2.12.

Do
Choose a low level heuristic uniformly at random and apply it to generate a new solution
Calculate Improvement
Accept new solution with probability
Update temperature
***Until** stopping condition is met.*

Figure 2.12: Simulated annealing hyper-heuristic (source from: Soubeiga 2003)

Simulated annealing hyper-heuristic was first applied to shelf allocation problem (Bai and Kendall 2005). Furthermore, Bai et al. (2007) solve three scheduling problems: nurse rostering, university course timetabling and one-dimensional bin packing problem by using this method. Stocastic heuristic selection was adopted to determine the capabilities of different heuristics

during the search. Simulated annealing acceptance criteria (*AM* and *OI*) is used to decide whether to accept a heuristic or not. Results showed that this method can produce competitive results. Another work on simulated annealing hyper-heuristic can be seen in Downsland et al. (2007).

2.4.5.6 Tabu Search Hyper-heuristic

In order to select which low level heuristic to apply at a decision point, tabu search based hyper-heuristic manipulates tabu list and adaptive memory to learn the performance and behavior of the low level heuristics. Tabu search hyper-heuristic keeps the non-domain specific information such as the number of heuristics, recent changes in the evaluation function, the CPU time required to implement the heuristic and the tabu duration.

Basically, the idea of this approach is that hyper-heuristic will rank a set of heuristics according to their performance in the search process. Similar to the tabu search concept, certain heuristics will be kept *tabu* at certain stage of the search process. At each iteration, the highest *non-tabu* will be the current solution until a termination condition is met. An example of pseudocode in their algorithm is as in Figure 2.13:

Construct initial solution
Do
Consider heuristics that are not tabu.
Apply chosen heuristic and make the heuristic tabu.
Update Solution.
***Until** terminating condition*

Figure 2.13: Tabu search hyper-heuristic (source from: Kendall and Mohd Hussin 2004b)

Kendall and Mohd Hussin (2004b) make *tabu* the list of heuristics that have been used too many times in order to permit other heuristics to have a fair opportunity to be selected by the hyper-heuristic. The *tabu* duration maintains how long a heuristic should stay in the list and will not be applied during the current iteration. For each iteration, the tabu duration value for each heuristic will be decremented until it reaches zero where the heuristic now is *tabu inactive*. The work was tested on eight examination datasets. For each dataset, different tabu durations and stopping conditions were investigated. Their result did not produce the best solutions, however, it can produce good feasible solutions and proved that tabu search based hyper-heuristic can be applied to different problem instances. For more work on tabu search hyper-heuristic can be seen in Hussin (2005).

Burke et al. (2003a) have successfully used the tabu search hyper-heuristic to solve the course timetabling and the nurse rostering problem. They make *tabu* the heuristics that do not improve the solution. The heuristics compete with each other by using rules based on reinforcement learning. A ranking mechanism dynamically sorts the low level heuristics used. The heuristic with the highest rank will then be applied. The non-performing heuristics will be moved to the *tabu* memory to prevent it from being chosen. For the nurse rostering problem as in Burke and Soubeiga (2003), the hyper-heuristic produced solution of acceptable quality and achieved optimality in some cases. Furthermore, in Burke et al. (2005c), the tabu search hyper-heuristic with fixed size of tabu list was used to solve the space allocation and timetabling problems and has produced results of acceptable quality.

2.4.5.7 Genetic Algorithm Hyper-heuristic

Genes in a chromosome represent low level heuristics in a Genetic algorithm (GA) hyper-heuristic. In Cowling et al. (2002a), a direct GA with a fixed length of chromosome is developed to solve the trainer scheduling problem. The low level heuristics contain a combination of operators such as adding, swapping and deleting events in the schedule. The crossover and mutation operators will randomly select some positions in one chromosome and mutate it to evolve new values. The new evolution of these procedures is a sequence of low level heuristics that will be applied to the problem in the sequence stated. In (Cowling et al. 2002b), an adaptive length of chromosome is further investigated to allow a dynamic removal and insertion of heuristics. This method was tested on a geographically distributed training staff and course timetabling.

In addition to adaptive length of chromosome in (Cowling et al. 2002b), a guided adaptive length of chromosome is applied to allow for efficient operation of dynamic removal and insertion of heuristics, thus evolving a quality sequence of heuristics to obtain a good solution. The work is tested on a student project presentation scheduling problem. Cowling et al. (2002d) perform a hyper-GA (an adaptive length chromosome) to a trainer scheduling problem. For each individual in the population encodes a sequence of low level heuristics to be applied and has proved to produce good results. The work was extended to introduce the dynamic length of the chromosome (Han et al. 2002). This is to allow for dynamic removal and insertion of heuristics which aim to find a good chromosome length. The work was applied to geographically distributed training staff and course timetabling problems and has produced good quality

solution. Furthermore, in order to enhance the efficiency of each gene in the chromosome, a *tabu* list is introduced (Han and Kendall 2003b). The role of the *tabu* list is to prevent the non performing low level heuristics to be applied in specific chromosomes (penalise the genes that do not change the objective function), thus improving the efficiency of each low level heuristics call. The results are improved in terms of computational time.

A sequence of dispatching-rules was presented as low level heuristics to solve the hybrid flow shop problem (Ochoa et al. 2009). Genetic algorithm was used as a high level search to search for two possible heuristics for each of six assignments. A landscape analysis was conducted in this work and concluded that search space of heuristics is effective to search solutions for production scheduling. In Terashima-Marin et al. (2006), a genetic-based method with a variable length representation was used to search for combinations of condition-action rules in order to produce a general hyper-heuristic to solve the two-dimensional cutting stock problems. The method was tested on large set of benchmark problems and has produced excellent results.

2.5 Ant Algorithm Hyper-heuristic

In an ant algorithm hyper-heuristic, a problem is represented as a directed graph where the nodes represent low level heuristics. Initially, ants travel the graph with initial solutions. To travel to the next node in the graph, an ant makes decision based on certain probability value. Once an ant arrives at a node (heuristic), it will apply the low level heuristic at the particular node of which a heuristic can be applied more than once. An example of a detailed algorithm can be obtained from O'Brien (2007). In the following section, a detailed procedure of this method is presented.

2.5.1 Choosing a Heuristic

This is the most important state in the algorithm as it chooses which heuristic to implement to produce a solution. A combination of pheromone and visibility determines the selection of a heuristic. In ant algorithm hyper-heuristic, pheromone value corresponds to the value of improvement produced by a heuristic performed by an ant (Burke et al. 2005b, Chen et al. 2007). A higher pheromone level indicates the confidence level for other ants to apply the same heuristics. Visibility value corresponds by the amount of *CPU* time a heuristic took to complete its task. Low CPU time to implement a heuristic is much preferred. In Chen (2006), the decision to move to another node (apply another heuristic) is based on the ant system transition rule (Dorigo et al. 1991). The probability of all nodes will then be summed up to generate a new node destination for the ants according to *Roulette Wheel Selection* algorithm. After visiting a node, the low level heuristics will be applied and a solution is obtained. Burke et al. (2003c) investigate the possibility of getting negative values for the pheromone and visibility values, thus calculating the positive values according to the *RouletteFunction.* More details of the calculation can be obtained from O'Brien (2007).

2.5.2 Pheromone Updates

As ants travel, they lay a chemical substance called pheromone (Dorigo et al. 1991). For both O'Brien (2007) and Chen (2006), the pheromone value is laid based on improvements made by the heuristics. O'Brien (2007) calculates the improvement as the difference between the best quality found during the current journey and the best quality found during the previous journey). To avoid the pheromone value goes unbounded, an *evaporation* process is applied where edges

corresponding to the heuristics will be punished by not receiving any pheromone. O'Brien (2007) further suggested that pheromone laid on the path after a certain number of heuristics is based on the improvement of the quality solution during a journey. A term '*cycle*' is used to identify the journey between all ants beginning their path and all ants completing their path. The algorithm can iterate for as many cycles as required. An edge is not given any pheromone value if it performs badly.

The amount of pheromone π_{ij} on each edge at time t as in O'Brien (2007) is as follows:

$$\pi_{ij}(t) = (1- \rho)\, \eta_{ij}(t - m.n) + \sum_{k}^{m} \frac{\#_{ij}(P_k(t)).I(P_k(t))}{T(P_k(t))}$$

where:

ρ	pheromone evaporation coefficient
m	number of ants in the colony
n	heuristic calls
$P_k(t)$	path ant k travel during the cycle ending at time t.
$\#_{ij}(P_k(t))$	number of times an edge (i,j) occurs during path $P_k(t)$
$I(P_k(t)$	improvement produced by heuristic ant k during its last path.
$T(P_k(t))$	CPU time taken

Good heuristics are always being rewarded with higher level of pheromone and this will lead to unsuccessful heuristic not being visited. Thus there is the potential of the algorithm to be trapped in the local optima. According to Chen et al. (2007):

"It is more important that the overall sequence of steps consisting of 'good' and 'bad' moves generate an appreciable improvement than to find individually 'good' moves.

Burke et al. (2003c) suggested that ants should not make any decision immediately after each move instead after the journey has been completed. They tested various lengths of journey and concluded that the length equal to the number of node (low level heuristics) is good. They implemented the same transition probability as in Dorigo et al. (1996) and with a new quantity of pheromone laid on edge (i,j):

$$\Delta\tau_{ij} = \frac{Q.I.N}{L_k}$$

where:

Q a constant

I total improvement of ant k over its journey

N number of times the edge was traversed.

L_k length of ant k's journey

Chen et al. (2007) applies the following formula to lay pheromone on their work:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{I_k}{L_k} + Q & \text{If } I_k > 0 \\ 0 & \text{Otherwise} \end{cases}$$

Where:

Q a constant

L_k length of ant k's journey

I_k total improvement of ant k over its journey

2.5.3 Visibility Updates

The visibility information together with the pheromone value helps in making the decision (probability calculation). Visibility in a TSP is the inverse value of the distance between node *i* to node *j*. However, in the algorithm, an ant has no information about each heuristic capability. Therefore, the visibility value is dynamic and continually adaptive. For example, the visibility value can be a value of how well heuristics work together or CPU time required to obtain a solution. It is updated each time ants visit a node (heuristic). O'Brien (2007) uses the choice function as in Soubeiga (2003) to determine the visibility function. The choice function was based on solo and sequential performance of the heuristics. The choice function used has included a diversification strategy which motivates the heuristics not recently used to be selected. The visibility function (η_j) is calculated as follows:

$$\eta_j(t) = \lambda\eta_j(t-m) + \sum_{k}^{m} \frac{I_{kj}(t)}{T_{kj}(t)}$$

where:

m number of ants in the colony

$I_{kj}(t)$ improvement produced by heuristic j on ant k's current solution at time t

$T_{kj}(t)$ amount of CPU time heuristic j took to run ant k's current solution at time t

λ a constant weight of recent performance with a value between 0 and 1.

Burke et al. (2003c) have suggested that the value visibility should be calculated as an inverse function of CPU time taken to implement a heuristic. However, Chen et al. (2007) have experimented that the visibility value as follows has resulted in a better solution.

$$\eta_{ij}(t) = \rho * \eta_{ij}(t-1) + \sum_{k}^{m} \frac{\lambda^{Ikj(t)}}{T_{kj}(t) * num(i,j)}$$

where:

ρ	constant between 0 and 1 to provide decay
$I_{kj}(t)$	improvement of ant k current solution
λ	a static number to convert negative value to positive value
$T_{kj}(t)$	amount of time needed to obtain a solution on heuristic j at time t
$num(i,j)$	number of edges an ant has travels.

Population of ants as hyper-heuristic agents constructed sequence of low level heuristics guided by the pheromone and visibility values. This has drawn some similarities with genetic algorithms hyper-heuristic (Cowling et al. 2002a) where a population of chromosomes is constructed and genes represent low level heuristics. Ant algorithm hyper-heuristic has been applied to a project presentation scheduling problem and travelling tournament problem (Burke et al. 2005b, 2003c) and travelling tournament problem (Chen et al. 2007), however it has never been applied to routing problems. This motivates the investigation of the hyper-heuristic approach based on ant algorithm to solve different routing problems.

2.6 Summary

We have reviewed some of the solution methods for solving optimisation problems in the literature. The best method to find optimal or good quality solution for optimisation problems is the exact methods. However, due to a large amount of computation time needed to solve large problems, heuristic methods are used. Heuristic methods are search technique that tries to find for good quality solutions at a reasonable computational cost. For the last twenty years, a more advanced technique which combines the basic heuristic methods with high level framework to efficiently search the search space was developed (Blum and Roli 2003). This technique is known as metaheuristics. It is considered as intelligent search methodologies. They have some form of learning mechanism to store information as the search progresses and can be combined with different concepts to explore the search space. We have presented a few metaheuristics methods in this chapter. Metaheuristics have been shown to work well on certain instances however, for other instances, they do does not perform well and often, they are expensive to adapt to new instances and problems. On the other hand, problem owners normally would prefer simple, easy to implement heuristics which do not require a large amount of resources and expertise for the development.

Therefore, hyper-heuristic methods were proposed to raise the generality of metaheuristics. The generality is demonstrated by selecting appropriate heuristics according to different problems and instances, utilising simple heuristics and allowing these heuristics to compensate each other's weaknesses. We have presented in this chapter several hyper-heuristic approaches and we

detailed the ant algorithm hyper-heuristic which provides a fundamental framework to our work throughout this thesis. In chapter 4, we shall introduce the ant-based hyper-heuristic and adapt it to routing problems. If ant-based hyper-heuristic produces good results in project presentation scheduling problem (Burke et al. 2005b) and the travelling tournament problem (Chen et al. 2007), we hypothesise that this algorithm will also work well on routing problems. We will present the routing problems (vehicle routing problems and travelling salesman problem) in the next chapter.

Chapter 3

The Vehicle Routing Problem and the Traveling Salesman Problem

3.0 Introduction

Among the most popular problems in routing problems are the vehicle routing problem (VRP) and the traveling salesman problem (TSP). The VRP involves in finding the lowest cost routes from a depot to a set of other cities or customers. The problem of TSP is to find a minimum length tour that visits each city exactly once and returns back to the starting city. This chapter discusses both the VRP and the TSP. Previous work on these problems is then reviewed.

3.1 The Vehicle Routing Problem (VRP)

The Vehicle Routing Problem (VRP) is a combinatorial problem (Cordeau et al. 2005) that was introduced by Dantzig and Ramser in 1959 (Dantzig and Ramser 1959). It has been largely researched because of its importance in the logistic and supply chains management. The VRP is the generalization of the traveling salesman problem (Pardalos et al. 2002).

Laporte (Laporte 2009) describes the VRP as:

"a problem of designing least-cost delivery routes from a depot to a set of geographically scattered customers, subject to side constraints".

The customers are referred as a stop or centre for delivery or pick up. Each customer will be served by only one vehicle. Example of real life applications includes the delivery of newspapers, the delivery of fresh goods or transportation of people. Each of these problems has its own set of constraints that have to be respected.

3.1.1 The VRP Variants

VRP consists of several variants. These variants are either constrained by the limit of capacity or on time windows. Limit on hours of a driver can work, the limit on length on route or limit on the service time a customer is served can create a variant. Among the variants are; the capacitated vehicle routing problem (CVRP), the VRP with time windows (VRPTW), dynamic VRP (DVRP), time dependent VRP variant (TDVRP) and VRP with pickup and delivery (VRPPD). We briefly describe these variants and discuss the CVRP in more detail as we will apply our work for this variant.

3.1.2 VRP with Time Windows (VRPTW)

VRPTW is associated with a time interval or time window. A general description of the problem is; there are vehicles with capacity q and each customer i with demand d_i. Each customers i has a time window $[a_i,b_i]$. The depot's time window is $[a_0,b_0]$, where a vehicle can only leave the depot after time a_0. The service for a customer i can start within a time window $[a_i,b_i]$ and a vehicle has to arrive at customer i before time b_i. An excellent overview of approaches tackled for VRPTW can be found in Braysy and Gendreau (2005a, 2005b).

3.1.3 Dynamic VRP (DVRP)

The basic VRP deals with customers with a known schedule in advance. This schedule includes the service time for each customer and the drivers' driving time. However, in real life application, this information often is undetermined or not known in advance. Furthermore, information can change after schedule has been produced. DVRP or online VRP will have to accommodate these new requests into an already planned schedule. A good survey on DVRP can be obtained in Gendreau and Potvin (1998).

3.1.4 Time Dependent VRP (TDVRP)

The objectives of this variant is to minimize the total time travelled and the number of tours travelled. Unlike many other variants, the travel time for TDVRP is dependent on time. The travel time between two customers is calculated between the points and the time of the day. There may be time window to constrain the time for serving the customers. The application of

this variant is common in many cities where the time travelled depends on the traffic that exists at the place. An optimisation procedure is normally performed to search for best starting times. Research done on TDVRP is presented in Ishoua et al. (2003) and Donati et al. (2003).

3.1.5 VRP with Pickup and Delivery (VRPPD)

In VRPPD, nodes in a graph can be represented either as pickup locations or delivery locations. Goods are required to be moved from these delivery or pickup locations. The objective of VRPPD is to find the optimal routes for the vehicles to visit these locations. Customers may have both delivery and pickup services. For example in grocery stores, reusable containers of certain goods need to be picked up and at the same time, goods have to be delivered. More information on VRPPD can be obtained from Dethloff (2001).

3.2 Capacitated Vehicle Routing Problem (CVRP)

CVRP is a *NP*-hard problem since it includes the TSP (Braysy et al. 2004). We can assume CVRP is related to two optimization problems; the TSP and the bin packing problem (BPP). CVRP is related to multiple TSP in such a way that if customers are assigned to a route, it is solved similar to solving the TSP i.e. to produce the shortest tour possible in a tour. The objective of BPP is to find the minimum number of bins required to solve a problem. VRP is similar to BPP in finding the minimum number of vehicles required to deliver demands for each customer.

The CVRP is a version of the vehicle routing problem (VRP) where a fleet of m vehicles with limited capacity Q has to visit a set of customers with specific demands. The sum of demands on any route should not exceed the vehicle capacity Q. The objective is to minimize the total travel distance. The CVRP can be formulated as follows: a connected graph $G = (V,E)$ of $n+1$ nodes, with a set of customers with demand q_i is represented as $v_i \in V$, $i = 1.....n$ with v_0 as the depot. The CVRP problem is to find a set of lowest cost vehicle routes so that the following constraints are satisfied:

1. Each vehicle's route starts and ends at the depot.
2. The total demand does not exceed the vehicle capacity Q for every route.
3. Each vertex except the first vertex (depot), v_0, is served exactly once by exactly one vehicle.

3.2.1 Approaches to Solve the Capacitated Vehicle Routing Problem (CVRP)

Various methods have been applied to solve the VRP. Survey on exact methods for VRP is described in Laporte (1992). Among the most efficient exact methods to solve the CVRP are the branch and cut methods. The earliest branch and cut methods were introduced by Christofides and Eilon (1969). Good results produced for CVRP with branch and cut methods are presented in Naddef and Rinaldi (2002), Ralps et al. (2003) and Lysgaard et al. (2004). We will not discuss in detail the exact approach for solution methods for CVRP. We will however, present some heuristics and metaheuristics approaches used to solve the CVRP.

3.2.1.1 Heuristics for CVRP

Heuristics method for CVRP produces feasible solutions at reasonable time. Laporte and Semet (2002) classify heuristics for CVRP into two categories; classical heuristics and metaheuristics. Classical heuristics were proposed forty years ago which include the saving heuristics (Clarke and Wright 1964) the sweep algorithms (Gillet and Miller 1974) and cluster-first route-second algorithm (Fisher and Jaikumar 1981). Metaheuristics include the tabu search (Osman 1993); simulated annealing (Osman 1993, Taillard 1993, Gendreau et al. 1994) and ant algorithms (Bell and McMullen 2004, Bin et al. 2009).

3.2.1.1.1 Saving Heuristics

The saving heuristic introduced by Clarke and Wright (1964) often produced relatively good solutions and normally is used to generate initial solutions. The savings are calculated by joining two routes into one route. The concept is illustrated in Figure 3(a) and Figure 3(b).

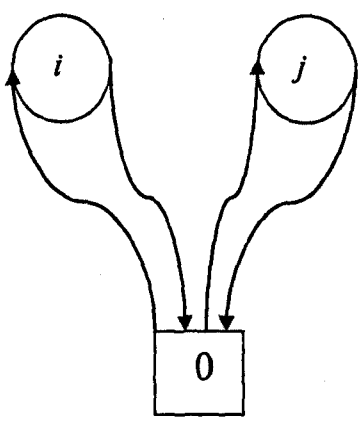

Figure 3.1(a): Initial route

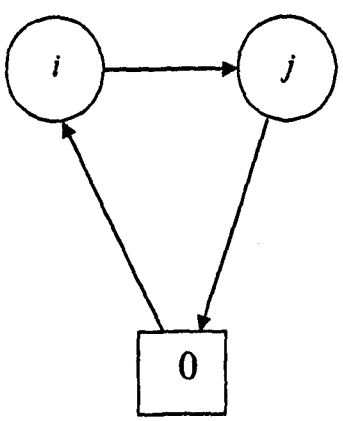

Figure 3.1(b): Alternative route

In Figure 3.1(a), customers i and j are initially visited on separate routes. As an alternative to this initial route, we can visit these two customers in one route as illustrated in Figure 3.1(b). The total cost of route in Figure 3.1(a) is calculated as:

$D_a = c_{0i} + c_{i0} + c_{0j} + c_{j0}$

And the total cost of route in Figure 3.1(b) is calculated as:

$D_b = c_{0i} + c_{ij} + c_{j0}$

By joining the two routes, the savings can be calculated as:

$S_{ij} = D_a - D_b$

These savings will then be sorted in decreasing order, customers i and j with the highest saving and without violating any constraint will be merged until no further merges are possible.

3.2.1.1.2 Sweep Algorithms

In this algorithm, an initial node is selected, and other nodes will be sorted accordingly to the angles from the initial node. The sweep algorithm introduced by Gillet and Miller (1974) is briefly described as follows:

Step 1: The polar coordinates of each node are calculated and sorted in increasing order.

Step 2: Starting from an initial node, feasible routes are created by rotating the angles starting from the smallest angle. The nodes are clustered in a vehicle route k such that the total capacity of each route does not exceed the capacity Q.

We will use this method to generate the initial solution for capacitated vehicle routing problem (CVRP) in this thesis.

3.2.1.1.3 Cluster-first Route-second Algorithms

In this algorithm, customers were first clustered into feasible routes; subsequently actual route will be constructed. Fisher and Jaikumar (1981) introduced the concept of this method by locating clusters based on some seeds. An example to select the criteria for these seeds is described in Baker and Sheasby (1999). The objective is to minimize the total distance in these clusters without violating any constraint. For each cluster, a route is constructed similar to solving the TSP.

3.2.1.2 Metaheuristics for CVRP

Metaheuristics for VRP were introduced last fifteen years ago. Compared to classical approach, metaheuristics allow wider search on the solution space, accepting low quality solutions and infeasible solutions. Several metaheuristics approaches will be discussed below.

3.2.1.2.1 Tabu Search

Tabu search is one of the popular approaches used to solve CVRP. A solution space is explored by moving from the current solution to the best solution found in the neighborhood. Solutions that were previously found will be kept *tabu* to avoid cycling for a number of iterations. Tabu search features such as short term memory where solutions posses certain attributes of current solution are declared *tabu* for a number of iterations to avoid cycling. However, if there is a new best solution among all known solutions that posseses the attribute found, the new solution is

accepted. Osman's algorithm in Osman (1993) used a fixed length of tabu tenure with no long term memory and intensification strategy. The work has produced significantly improved results on benchmark datasets. The taburoute algorithm introduced by Gendreau et al. (1994) utilizes a continuous diversification scheme and two types of intensification scheme. A Taillard algorithm which has produced 12 best known results out of 14 CMT instances is the best tabu search heuristics for CVRP (Taillard 1993). It employs a decomposition scheme that utilizes the parallel computing processors. Customers are partitioned into sectors centered at the depot. Solutions are searched within these sectors by different processors. A detailed description on tabu search applied for VRP is provided in Cordeau and Laporte (2005).

3.2.1.2.2 Simulated Annealing

In a simulated annealing, non-improvement solutions will be accepted with certain probabilities, which are determined by a control parameter (T), a temperature that is controlled by a deterministic cooling schedule. The simulated annealing algorithm for CVRP implemented in Osman (1993) utilizes the non-monotonic cooling schedule which requires specific information such as starting and final temperature, a decrement rule to updated the temperature after each iteration, an update rule to reset the parameter if the system freezes and a stopping condition. This method implements the I-interchange moves (Osman 1993) to generate new solutions. The algorithm is tested on 17 benchmark problems from the literature and produces 10 new best solutions.

3.2.1.2.3 Ant Algorithms

In ant algorithms (Bell and McMullen 2004), a colony of ants is used to construct routes for CVRP. An individual ant acts as a vehicle and will select customers starting from the depot until all customers are selected. It will return back to the depot when the capacity constraint is met. To select a customer, the ant follows a probabilistic rule and to improve the selection of customers in next iteration, the pheromone trail will be updated based on the previous performance. A value to control the evaporation rate is given. After a number of iteration, a global updating rule on the pheromone value is enforced on the trails. This approach has produced good results (1% of best known results) on small instances, however for larger instances, the results are not efficient. An improved version of this algorithm (Bin et al. 2009) introduced a new method to update the pheromone trail (ant-weight strategy) and mutation operators to solve the CVRP. The new ant-weight strategy updates the pheromone trail by combining both the global pheromone increment and the local pheromone increment. The global increment solution is related to the total length of the solution and the local pheromone is related to the contribution of the specific trail to the solution. The mutation operators applied mimic the genetic algorithm which alters the nodes (customers) at a predefined probability. This is to reach further solutions in the search space. Results produced from this algorithm are competitive with the results in the literature.

3.3 The Traveling Salesman Problem (TSP)

The TSP involves finding the shortest tour in a route; searching for an order in which each city should be visited, starting from the first city, visiting each city exactly once and returning to the starting city. The history of TSP was believed to be found in 1920 in Vienna (Applegate et al. 1998). Later in 1954, a formal description of TSP was formulated (Dantzig et al. 1954). The TSP is involved in many applications in real life. A simple example is the delivery service where a postman would like to find the shortest route to cover his daily task. The objective of the TSP is to find a minimum length tour that visits each city exactly once and returns back to the starting city (Gutin and Punnen 2002). Mathematically, it can be represented as adapted from Helsgaun (2000) as follows:

Given a cost matrix $C = (c_{ij})$ where c_{ij} represents the cost of travelling from city i to city j. We will find a permutation $(i_1, i_2, i_3, \ldots i_n)$ of the integers from 1 through n that minimizes the quantity $c_{i1i2} + c_{i2i3} + \ldots + c_{ini1}$

The properties of the cost matrix can be defined as follows:

- If $c_{ij} = c_{ji}$ for all i and j, the problem is said to be a symmetric problem.
- If the triangle inequality holds ($c_{ik} \leq c_{ij} + c_{jk}$ for all i, j, and k) the problem is said to be metric (that is for any cities a, b, c, the distance between a and c must be at most the distance from a to b plus the distance from b to c).
- If c_{ij} are points in a plane geometry, it is a Euclidean problem.

TSP has been solved using numerous methods. It can be categorized into two; the tour construction and tour improvement. These methods will be discussed as below.

3.3.1 Tour Constructions

A constructive heuristic builds a tour from scratch. The algorithm stops when a solution is found and no attempt is made to improve the solution. Examples of constructive heuristic approaches for the TSP can be seen in a paper by Bentley (1992).

3.3.1.1 The Nearest Neighbour

The nearest neighbor method is considered as one of the simplest and straightforward TSP heuristics. The method visits the nearest city that has not been visited yet. Once all cities have been visited, the tour is completed by returning to the starting city.

The algorithm, more formally, is as follows:

1. Select a random city.
2. Find the nearest unvisited city and move to it.
3. If there are any invisited cities left, repeat step 2.
4. Return to the first city.

3.3.1.2 Insertion Heuristics

A basic insertion heuristic starts with a tour of a subtour of all cities, then inserts the remaining cities utilising some heuristics. The initial subtour is often a triangle or a convex hull. The tour can also start with a single edge as a subtour.

The algorithm is as follows:

1. Select the shortest edges and make a subtour of it.
2. Select a city that is not in the subtour, that is the shortest distance to any one of the cities in the subtour.
3. Find an edge in the subtour where the cost of inserting the selecting city is minimal between the edges.
4. Repeat step 2 to step 3 until all cities have been inserted.

The set of cities which make up the starting subtour is usually chosen at random.

3.3.2 Tour Improvement

Once a tour has been constructed, we can improve the tour using tour-improvement heuristics. Among the most common for the TSP are 2-opt, 3-opt and the generalisation to *k*-opt moves and the Lin-Kernighan heuristic.

3.3.2.1 The 2-opt, 3-opt and *k*-opt Move

The 2-opt move removes two edges from a tour, breaking it into two segments. The tour is then reconstructed by re-joining the two segments. There is only one way (apart from the original tour) to reconnect the two segments so that it will produce a valid tour. This will be done if the new tour is shorter than the current tour. This process will be repeated (removing and reconstructing) until no further 2-opt improvements can be found and the tour is now 2-optimal. Figure 3.2 illutrates the 2-opt moves for a symmetric TSP. A 2-opt move represents an improvement to the current tour if $d(a,b) + d(a_1,b_1) < d(a,a) + d(b,b_1)$.

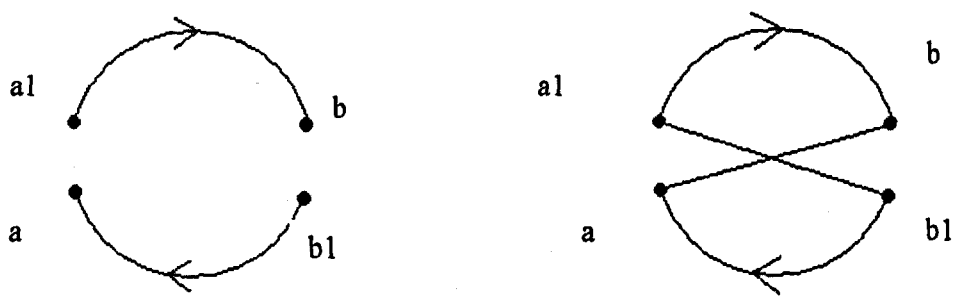

Figure 3.2: 2-opt Moves

The 3-opt move is an extension of 2-opt. Instead of removing two edges, we remove three edges from the tour. An example to illustrate the moves is shown in Figure 3.3. To form a valid tour, there are two ways of reconnecting the three paths. A 3-opt move can also be seen as two or three 2-opt moves. This process will be repeated (removing and reconstructing the tour) until no further 3-opt improvements are found and the tour is now 3-optimal and also 2-optimal.

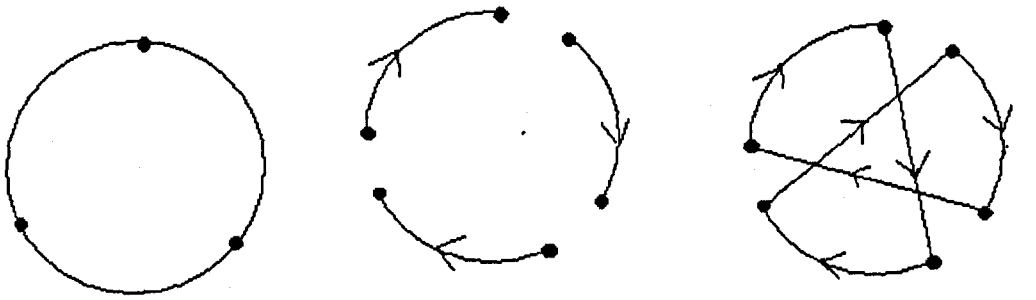

Figure 3.3: 3-opt Moves

An extension of 3-opt is the 4-opt or the *k*-opt move. As *k* increases the moves will take more time.

3.3.2.2 Lin-Kernighan Heuristic

The Lin-Kernighan heuristic is based on a generalisation of *k*-opt moves. It is known to be an effective method to obtain optimal and near-optimal solutions for the symmetric travelling salesman problem. It decides which *k* is the most suitable at each iteration step.

Lin-Kernighan neighborhood moves are based on the observation that a *k*-opt move can be constructed sequentially by performing a sequence of tour reversals such as 2-opt moves. The search space can be reduced significantly by considering only moves with positive gains. For a more in-depth study of the Lin-Kernighan heuristic, see Lin and Kernighan (1973).

3.4 Summary

The study on VRP and TSP has been researched for more than 50 years ago. We present in this chapter several solution methods for both problems. For VRP, these include the heuristics and metaheuristics methods used to solve the problem. For TSP, the solution methods presented are heuristics for tour construction and tour improvement.

Chapter 4

Ant-based Hyper-heuristic

4.0 Introduction

Our aim in this thesis is to develop a method that can be applied to many routing problems or problem instances. We have discussed the concept of hyper-heuristic in chapter 2. In section 2.5, two ant algorithm hyper-heuristic (Burke et al. 2003c, Chen et al. 2007) were presented. The framework of these approaches is observed and the drawbacks are analysed to form a basis for our general approach in solving the routing problems.

In this chapter we present two descriptions for the ant-based hyper-heuristic. The general design of the proposed approach is presented in section 4.1. In section 4.2, the ant-based hyper-heuristic is described by examining the ant system algorithm (Dorigo et al. 1996). In ant system algorithm, the key properties of these approaches are the pheromone and heuristic information (visibility) updating activities. We investigate how to present this information and address the comparison between the ant system algorithm and our approach. Furthermore, we present the similarities and differences of our approach to another ant algorithm hyper-heuristic as in Burke et al. (2003c) and Chen et al. (2007).

The key process of any ant algorithm is the pheromone and visibility updating rules. We observe these procedures and propose new updating rules. Secondly, in section 4.8, to further expand our work, a new updating rule for the pheromone rule is presented to compare the effectiveness.

4.1 Design issues

Soubiega (2003) was the first to address the design of hyper-heuristic methods. To select low level heuristics, several approaches can be used to guide the hyper-heuristic. If no learning mechanism is needed, selection will be based on random selection. However to perform an intelligent selection, some learning mechanism is required. In our work, we use ant-based hyper-heuristic as our learning mechanism. So far, we are aware that there are only two previous works which have investigated the ant-based hyper-heuristic (Burke et al. 2005, Chen et al. 2007).

In an ant algorithm, a given problem is represented as directed graph and the set of nodes is the candidate's solution to a given problem. For example in the TSP, the node represents the candidate city for the salesman to visit. Each edge has an associated distance (visibility) from city *i* to city *j*. Several ants are needed to perform a tour in order to achieve the objective function (that is to find the shortest route). These ants will traverse the edges guided by some information; the pheromone value which is a chemical substance that was left by the ant after it has performed a tour and the distance between cities *i* to city *j* is the visibility (heuristic information) that helps the ant to make decisions. More detailed explanation on how the ant algorithm work can be seen in section 2.3.6. Figure 4.1 illustrate the concept.

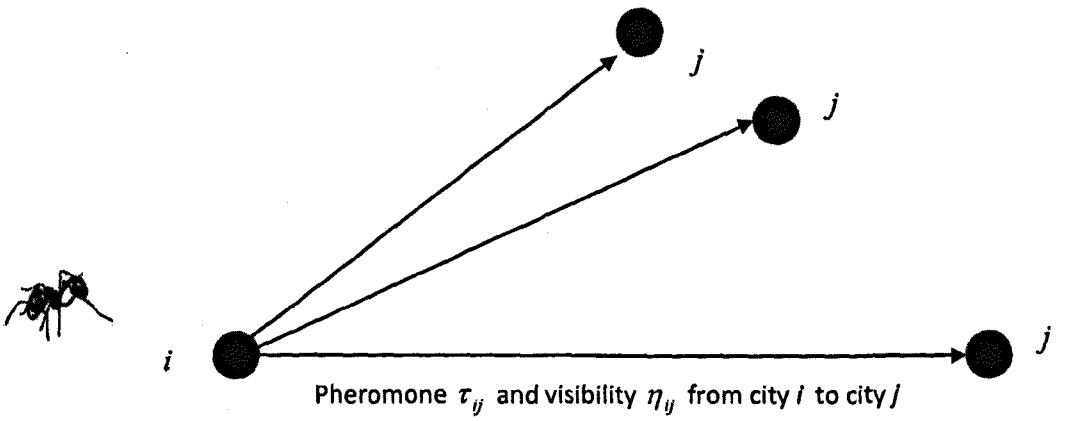

Figure 4.1: Ant arrives at city i, will choose to visit next city j based on the function of pheromone values τ_{ij} and heuristic values η_{ij}

We adopt the general guideline for hyper-heuristic design as in (Soubeiga 2003). This is presented in Figure 4.2.

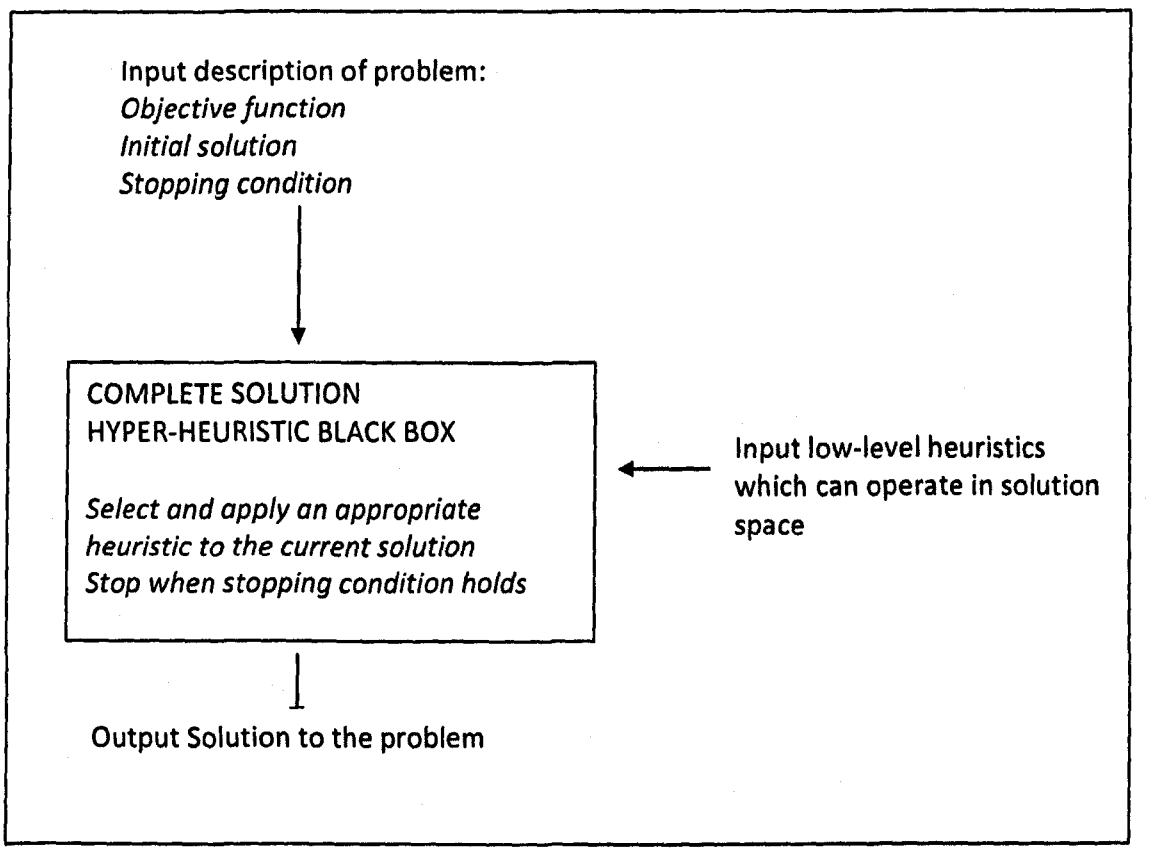

Figure 4.2: The general hyper-heuristic framework

The necessary input for this framework includes:

1. ***The objective function*** – is a measure of quality for a solution. In our work, we represent the objective function of each problem as an input to the hyper-heuristic framework.

2. ***Initial solution*** – in our work, we choose to deal with complete solutions. In the beginning of the algorithm, ants carried with them the initial solution. These initial solutions are constructed by some heuristic methods of which the quality is often poor but fast to construct.

3. ***Stopping condition*** – stopping condition is determined and input by the user. Usually, the stopping condition can be expressed as a number of iterations or the *cpu* time allocated for the execution of the algorithm. In our work, we set the stopping condition when there is no improvement in the solutions after certain number of iterations (for example no improvement after 100 iterations). This is in order to reflect the generality of the algorithm.

4. ***Low level heuristics*** – Low level heuristics are simple neighbourhood moves or simple heuristics that are problem dependent used to solve a specific problem. Hyper-heuristic communicates with low level heuristics based on non-domain specific information such as *cpu* time and the objective function passed over by these low level heuristics. Soubeiga (2003) addressed issues regarding the number of low level heuristics that a hyper-heuristic framework should employe in a certain study. He concludes that if the number of low level heuristics is too few, there will be no point in employing a hyper-heuristic approach. However, if there were too many low

level heuristics used, it will be computationally expensive. We will carry out experiments to investigate how many low level heuristics are appropriate in chapter 5.

4.2 Ant System Algorithms

In this section, we present in detail the ant system algorithm. It is important that this section is included in this chapter as it is the basis of our work for this thesis. An ant system was the first algorithm to illustrate the bahaviour of the artificial ants (Colorni et al. 1992, Dorigo et al. 1991, 1996, Costa and Hertz 1997). It was initially applied to the TSP and it works as follows:

A number of m ants are initially placed randomly on n nodes. It is usually the case that $m = n$, where the number of ants is set equal to the number of nodes (cities). The ant builds a tour incrementally by applying a state transition rule. A pheromone value, which is a desirability measure between each node, will be maintained by the ants. Once all ants have completed a tour, the pheromone value will be updated. Ants prefer to move to nodes which are connected by short edges (often referred to as visibility) and have a high amount of pheromone (i.e. more desirable), and there is a parameters setting (α and β) to find a balance between the pheromone values on the edges and the visibility.

In order to avoid unlimited accumulation of pheromone, an evaporation mechanism is added. This is implemented by multiplying the pheromone by a value between 0 and 1 at given times in the algorithm. This results in the desirability of edges becoming reduced if no new pheromone is added. More formally the algorithm can be described as follows. Let $\tau_{ij}(t)$ be the intensity of

pheromone trail on edge (i, j) at time t. At time t, each of the ants will choose to move to another node. Once the ants complete a tour (i.e. visited every city), this is regarded as one iteration.

The probability ρ_{ij} of an ant moving from node i to node j, at time t, is given by the following formula (Dorigo et al. 1996):

$$\rho_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{j \in allowed} [\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}} & \text{if } j \in allowed \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where :

τ_{ij} value of pheromone from node i to node j

η_{ij} heuristic information (this is given by the inverse distance, $1/d_{ij}$, where d_{ij} is the distance between node i and node j)

α the relative influence of pheromone from node i to node j

β the relative influence of heuristic information ($1/d_{ij}$) from node i to j

Ants will continue visiting cities until a tour is complete. Once a tour is generated, the length of the tour is calculated and the best length tour is recorded. For each tour, each ant k will deposit a quantity of pheromone value. This pheromone updating mechanism allows a greater amount of pheromone to be laid on shorter tours. The formula for trail updating τ_{ij} is as follows:

$$\tau_{ij}(t+1) = \rho.\tau_{ij}(t) + \Delta\tau_{ij}(t, t+1) \qquad (2)$$

where $\Delta\tau_{ij}(t, t+1) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t, t+1)$

The ρ value indicates coefficient (1 - ρ) which represents the evaporation of pheromone trail laid on edge (i,j) between time (t) and ($t+1$). The $\Delta\tau_{ij}(t, t+1)$ is a pheromone value laid on edge (i,j) by k-th ant between time t and $t+1$.

In TSP a city can only be visited once. Therefore to satisfy the contraint, a data structure (tabu list) is maintained. The tabu list saves the cities that has already been visited up to time t and forbids the ants from visiting them again before n iterations have been completed. After a tour is completed, the tabu list will be emptied and the ant's current solution (distance travelled) is computed. Visibility η_{ij} is a heuristic information which is an inverse of the distance ($1/d_{ij}$) between city i and j. In ant system, the visibility information remains the same throughout the algorithm.

4.3 Methodology

In ant-based hyper-heuristic framework, the search space is represented as directed graph where the nodes represent low level heuristics. Each edge will have an associated weight. In this environment, we will assume the edge will represent non-domain specific information. There are actually two edges between each node, representing the pheromone and the visibility values. Figure 4.3 illustrate the network.

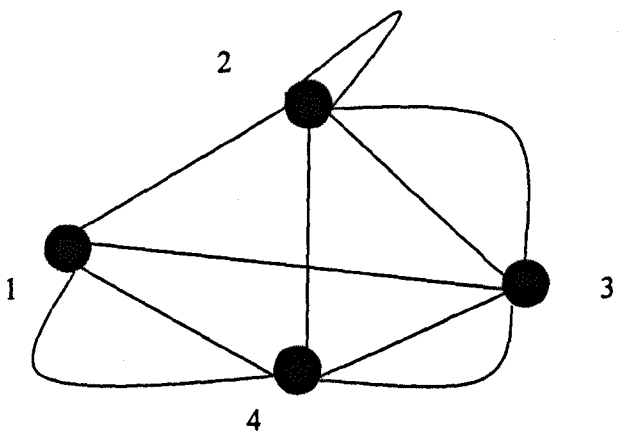

Figure 4.3: Fully connected graph V (the search space) with set of nodes E (set of low level heuristics)

In Table 4.1, we present the comparison between the ant system algorithm and our ant-based hyper-heuristic. For the ant system, the TSP is taken as a problem domain to illustrate the comparison.

Difference	Description	Ant system (AS)	Ant-based hyper-heuristic
α	Relative influence of pheromone trail	1 (this parameter has resulted in good performance for AS (Dorigo and Stutzle 2004)	We carried out experiments to investigate the best value of α see section 5.6.2
β	Relative influence of visibility (heuristic information)	2 – 5 (this parameter has resulted in good performance for AS (Dorigo and Stutzle 2004)	We carried out experiments to investigate the best value of β see section 5.6.2
Number of ants	Ants are agents to search for good solutions.	The number of ants is always set equal to the number of cities	The number of ants is set equal to the number of heuristics available
Initial placement of ants	Initial placement is where the ants are placed at the starting of the algorithm	Initially, ants are placed randomly on the cities available	We carried out experiments to investigate the best initial placement of the ants, see section 5.6.1
Choices of nodes	Ant choose to move to one node after another in a tour	Ant chooses to move based on probability transition rule which is a function of the visibility (distance) and pheromone value	Ant chooses to move based on probability transition rule which is a function of the visibility (*cpu* time) and pheromone value (based on the improvement made). Detailed information of this procedure is shown in section 5.6.2
Pheromone update/deposit	An activity to determine how much the pheromone value should be left on a particular path	Pheromone trails are updated after all ants have constructed their tours. In this context, a tour is considered when an ant has completed visiting all cities	Pheromone trails are updated after all ants have constructed their tours (in this context, a tour is equivalent to the number of low level heuristics available in the algorithm)

		available in the problem	
Visibility (heuristic information)update	An activity to determine what is visibility and how it is updated during the execution of the algorithm	The distance between city *i* and city *j*. These values remain static throughout the algorithm	The *cpu* time between heuristic *i* and heuristic *j*. These values are dynamic, changing as the algorithm progresses
Visited nodes	Already visited nodes	Ants are prohibited to make a visit to already visited cities until a tour is completed	A node (heuristic) is allowed to be visited as many times as possible
Evaporation rate (ρ)	An activity to decrease the value of pheromone so that it does not grows unbounded	$\rho = 0.5$	We carried out experiment to investigate the best value of ρ see section 5.6.3

Table 4.1: The comparison between the ant system and ant-based hyper-heuristic

Initially, ants will be placed at a node on the search space (the directed graph). Our ant-based hyper-heuristic uses the same idea as in Chen (2006). The nodes on the graph represent the low level heuristics. The number of ants used in this experiment is set to be equal to the number of nodes in the network and each ant carries an initial solution. Each ant will perform a tour by visiting a sequence of nodes by selecting a node being guided by the pheromone and visibility values. Once an ant arrives at a particular node, it will apply the low level heuristic to its solution. The solutions are continuously modified whenever an ant arrives at a new node. Unlike the TSP, a node is allowed to be visited several times during a tour. After a given number of tours, the ants will report the best solution found. The methodology is illustrated below.

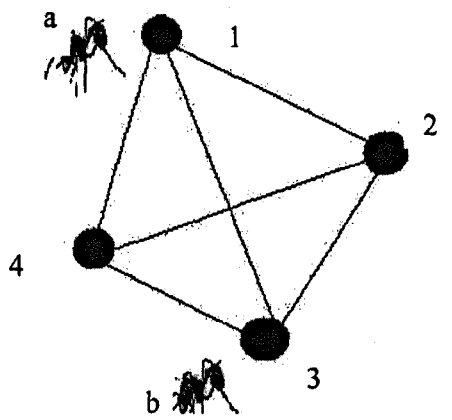

Figure 4.4(a): initial placement

i) There is a network with four nodes (low level heuristics). Initially two ants are placed on any of the nodes as a starting position. They carry with them an initial solution.

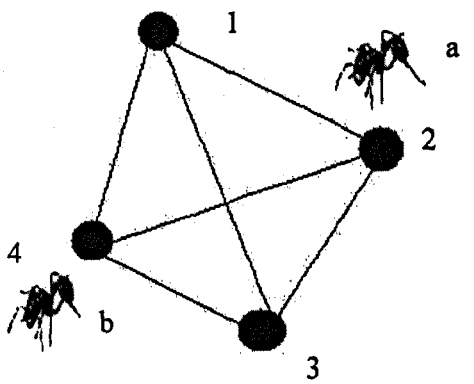

Figure 4.4(b): heuristics selection

ii) With probability transition rule (eq.1) the ants choose the next heuristic to visit. Ant *a* has chosen to move to heuristic 2 and ant *b* has chosen to move to heuristic 4. When they arrive at their destination, the ants will apply the heuristic to their solution. If the current solution obtained is better than the previous solution, it will replace the solution carried by the ants. These new solution will be carried to the next iteration.

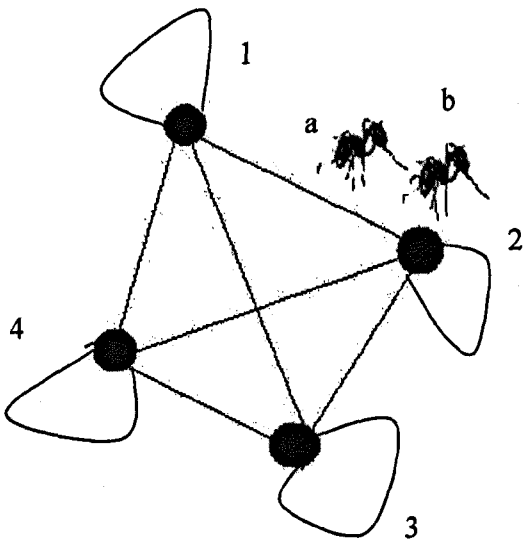

Figure 4.4(c): ants apply heuristics

iii) For the next iteration, these ants will traverse the nodes. For example, ant *a* has chosen to remain at the same node (heuristic 2) and will apply this heuristic. An ant can stay (or return) at the same heuristic. Ant *b* has chosen to move to heuristic 2 and applies the heuristic to modify its current solution.

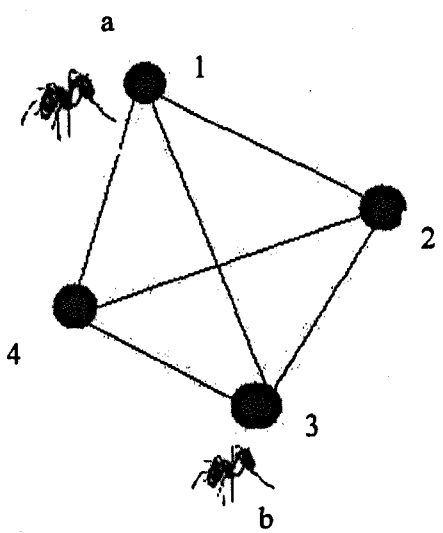

Figure 4.4(d): ants complete their tour

iv) A tour is considered when an ant has completed its journey through several sequences of heuristics. For this study, we consider a tour to be equivalent to the number of heuristics available. The stopping condition is set if no improvement is made for 100 iterations. The best solution will be reported when the stopping condition is met.

In Table 4.2 and Table 4.3, we further present the similarities and differences between the previous approach of ant algorithm hyper-heuristic in Burke et al. (2003c) and Chen et al. (2007) and our approach. We run experiments with these algorithms and the results will be presented in the next chapter. Detailed explanation of our approach will be described in the following sections.

Activities	i. Burke et al. (2003c)	ii. Chen et al. (2007)	iii. Ant-based hyper-heuristic	iv. Ant colony hyper-heuristic
Function	Explore search space of solution	Explore search space of solution	Explore search space of solution	Explore search space of solution
Low level heuristics	Vertices represent low level heuristics	Vertices represent low level heuristics	Vertices represent low level heuristics	Vertices represent low level heuristics
Heuristic selection: Each decision point, ants select next vertex to visit, apply heuristic on current solution	see section 2.5.1	see section 2.5.1	see section 4.5	see section 4.5

Table 4.2: The similarities of other ant algorithm hyper-heuristic (i and ii) to our approach (iii and iv)

Activities	i. Burke et al. (2003c)	ii. Chen et al. (2007)	iii. Ant-based hyper-heuristic	iv. Ant colony hyper-heuristic
Problem domain	1. Project presentation scheduling problem 2. Travelling tournament problem	1. Travelling tournament problem	1. Vehicle routing problem 2. Traveling salesman problem	1. Vehicle routing problem 2. Traveling salesman problem
Number of ants	Not specified	Not specified	Ants are set to be the same number of heuristics available in the network	Ants are set to be the same number of heuristics available in the network
Pheromone updates: After ants have visited a certain number of heuristics, they pause to analyse the path and lay amount of pheromone according on improvement on the quality of the solution	Only edges that contribute to improvement to current solution are rewarded with pheromone. (see section 2.5.2)	Only edges that contribute to improvement to current solution are rewarded with pheromone (see section 2.5.2)	All visited edges will be given some amount of pheromone. The distribution of pheromone values will be distributed proportioned to the performance done by the ants (see section 4.6)	There are two procedures involved; the local and global update. The global update will reward edges that produce the best solution. Local update is performed after each ant performs a tour. (see section 4.8)
Visibility updates : Visibility is continuously adaptive since it does not have the knowledge of heuristic's potential.	experimented the visibility value as in section 2.5.3.	experimented the visibility value as in section 2.5.3.	measured as computational time taken for a heuristic to complete its task (see section 4.7)	measured as computational time taken for a heuristic to complete its task (see section 4.7)

Table 4.3: The differences of other ant algorithm hyper-heuristic (i and ii) to our approach (iii and iv)

4.4 Initial Setup

A tour counter t is set to the same number of low level heuristics available. A tour is considered as to how many heuristics are allowed for an ant to finish its tour. We set the terminating condition to 1000 iterations. However, if the algorithm does not meet any improvement for 100 iterations, it will be terminated and the best result will be returned. To determine the appropriate setting for visibility value η_{ij} and the pheromone τ_{ij} value, an experiment is conducted in following chapter (section 5.6.2). Initially, each ant k is supplied with an initial solution S_k. S_b which carries the best solution is set to null. Pheromone evaporation rate is an important activity to avoid the pheromone values grows unbounded. We conduct experiments to determine the evaporation rate (section 5.6.3). We follow the ant system as in Dorigo (1991), where the number of ants is set equal to the number of heuristics. We experiment the best initial placement of the ants in chapter 5 (section 5.6.1).

4.5 Choosing a Heuristic

An important issue concerning the design of our hyper-heuristic is how the ants make decisions as to which heuristic to visit next. The decision will be made based on the probability transition rule as defined in the Ant System (Dorigo et al. 1996), but suitably adjusted for our hyper-heuristic framework. The probability transition rule as defined in the Ant System is shown (eq. 1).

In this thesis, we follow (eq. 1), however the representation of each parameter is adjusted to our framework. The probability transition rule refers to the probability of ant k choosing heuristic j from heuristic i. This probability function combines the value of pheromone τ_{ij} that exists on path (i,j) at time t and visibility η is the heuristic information of the quality of heuristic j. The α value represents the weight for the pheromone value and β value represents the weight for the heuristic information. The α and β parameters define the relative importance of these values. In our work, a heuristic is allowed to be visited more than once however in (eq. 1), a node is not allowed to be visited more than once, and thus we define the representation as follows:

$$\rho_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum [\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}} \end{cases} \qquad (3)$$

τ_{ij} value of pheromone from heuristic i to heuristic j

η_{ij} heuristic information (this is given by *cpu* time between heuristic i and heuristic j)

α the relative influence of pheromone from heuristic i to heuristic j

β the relative influence of heuristic information from heuristic i to heuristic j.

4.6 Pheromone Updates

As ants travel, they deposit a chemical substance called pheromone. After all ants have completed their tour, they will update the pheromone values. In a standard Ant System, the amount of pheromone corresponds to the quality of the solution found by the ants. In this respect, the hyper-heuristic algorithm mirrors the original ant algorithm. In our framework, the amount of pheromone corresponds to the quality of the solution found by the ants, as such that the quality is considered as improvement value between previous solutions and current solutions made by each ant in a tour. A tour for each ant is considered when the ant has completed its visit to several sequences of heuristics on the network. Once a tour has been completed by an ant, the improvement of the entire tour is computed. In the previous ant hyper-heuristic (Burke et al. 2003c and Chen et al. 2007), only edges that contribute to improvements to the current solution are rewarded with pheromone values. However, in our approach, we are concerned with giving all the edges visited with some amount of pheromone, regardless the edges contributed to improvements or vice versa. We believe that poor-performing visited edges might produce better solutions in the future and therefore should be considered in the decision making. The distribution of pheromone values will be distributed proportioned to the performance done by the ants. The pheromone deposited on each edge is calculated as follows:

$$\Delta\tau_{ij}^{k}(t) = \frac{I_{kj}(t)}{\Sigma I_{kj}(t)} \qquad (4)$$

where:

$\tau_{ij}^{k}(t)$ value of pheromone from heuristics i to heuristic j at time t

$I_{kj}(t)$ improvement made by ant k on heuristic j at time t

An example to determine the value of pheromone left on the edge is presented below:

There are improvements made by three ants, ant 1 = 154.17, ant 2 = -130.76, ant 3 = 76.67; we take the minimum absolute value of these improvements, ant 2 = abs(-130.76) + 1 and normalize all the values; ant 1 = 285.93, ant 2 = 1, ant 3 = 208.43. We then compute the total summation of these improvements, total = (154.17 + 130.76 + 76.67 = 495.36). The pheromone value is then calculated: ant 1: 285.93/495.36 = 0.577217, ant 2: 1/495.36 = 0.002019, ant 2: 208.43/495.36 = 0.420765. This is simplified in Table 4.4.

Ant	Improvement between previous solution and current solution of ant k, $I_{kj}(t)$	Normalised values of the improvements	Value of pheromone left on the edge $I_{kj}(t)/\sum I_{kj}(t)$
Ant 1	154.17	285.93	0.578
Ant 2	-130.76	1	0.002
Ant 3	76.67	208.43	0.420

Table 4.4: Value of pheromone left on the edge

These values measure the pheromone quality which indicates that higher improvements represent higher pheromone levels while smaller improvements represent smaller pheromone values. These values are then used in (eq. 5) to increase the pheromone τ_{ij} on each edge of the tour.

$$\tau_{ij}(t+1) = \rho.\tau_{ij}(t) + \Delta\tau_{ij}(t, t+1) \qquad (5)$$

where $\Delta\tau_{ij}(t, t+1) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t, t+1)$

The ρ value indicates a coefficient (1 - ρ) which represents the evaporation of pheromone trail laid on edge (i,j) between time (t) and ($t+1$). The $\Delta\tau_{ij}(t, t+1)$ is a pheromone value left on edge (i,j) by k-th ant between time t and $t+1$.

4.7 Visibility Updates

In addition, we consider the visibility information which represents some forms of heuristic information, which is combined with the pheromone value in order to decide which heuristic to visit next. In an ant system algorithm (Dorigo et al. 1996), the heuristic information was the distance between two cities (as it was tackling the travelling salesman problem). As such, it remained static throughout the algorithm. In our algorithm we consider the heuristic information by how well two heuristics work together. This is measured as the computational time and, as such, it is now a dynamic value over the course of the algorithm.

$$\eta_{ij} = 1/cpu\ time \tag{6}$$

where the *cpu* time is measured in seconds.

Figure 4.5 presents the ant-based hyper-heuristic that we developed.

1: $t = 0$ — *t is the tour counter. A tour is equivalent to the number of low level heuristics available in the algorithm.*

2: *counter* = 0 — *user defined counter for stopping condition*

3: $\tau_{ij}(t) = c$; $\Delta \tau_{ij} = 0$ — *c is an initial value of pheromone value* τ_{ij} *given for every edge (i,j)*

4: $\eta_{ij} = 1$ — *visibility* η_{ij} *is initially given value 1 for all heuristics*

5: Set S_k and S_b as null — S_k *is each ant's solution,* S_b *is best solution*

6: Place m ants on n heuristics

7: For $k = 1$ to m — *place the starting node of the ants.*
  Repeat until ants complete a journey
    For $k = 1$ to m do
      Select the next heuristic (node j) to apply based on pheromone and visibility
      Move the k-th ant to heuristic j
      Apply heuristic j to S_k to obtain new solution S'_k
      If $(S'_k < S_b)$
        $S_b = S'_k$
      Update visibility η_{ij} — *visibility is updated after a heuristic is applied*
      $t = t + 1$
    end for
  counter ++

8: If (t = = n) — *a tour for all ants is completed*
    Update pheromone value τ_{ij}
    counter = 0
    If *(any best solution is in this cycle)*
      For all ants $S_k = S_b$
    Endif
  Endif

9: If ($t = t_{max}$)
  Output best solution S_b found
  Stop

Figure 4.5: Overview of ant-based hyper-heuristic

4.8 Ant Colony (ACO) Hyper-heuristic

In this section, we describe the second framework for our ant-based hyper-heuristic. We developed the ant colony hyper-heuristic (ACO hyper-heuristic). The idea behind this approach is obtained from Dorigo and Di Caro (1999b; 1999c). Our ACO hyper-heuristic applies the same methodology as the ant-based hyper-heuristic; however it differs from the above methodology in the updating of the pheromone trail procedure. There are two procedures involved; the local and global update. The global update will use the best solution found at the current iteration to update the pheromone trail. This action is intended to reward the path that produces the best solution. The path $(i^* j^*)$ that belongs to the best solution S_b found by m ants will be updated with the new pheromone values given by the following equation:

$$\tau_{i^* j^*} = (1 - \alpha).\ \tau_{i^* j^*} + \alpha\ (L)^{-1} \qquad (7)$$

where:

$\tau_{i^* j^*}$	pheromone value for the path $(i^* j^*)$ that belongs to the best solution S_b
α	the evaporation rate for the pheromone
L	the total solution for the tour

The motivation for global update procedure is to encourage the use of good-performing edge and increase the probability of choosing the specific path. Local update is performed after each ant performs a tour. The procedure is similar to pheromone updates as in section 4.6. This is to simulate the evaporation procedure in order to ensure that the pheromone values do not go unbounded. The following equation is used:

$$\tau_{ij} = (1 - \alpha).\tau_{ij} + (\alpha).\tau_0 \qquad (8)$$

where:

τ_{ij} pheromone value that exists on the particular path $(i\,j)$

α the evaporation rate for the pheromone

τ_0 initial pheromone value assigned to all paths in the network

Figure 4.6 presents the ACO hyper-heuristic that we develop.

1: $t = 0$ — *t is the tour counter. A tour is equivalent to the number of low level heuristics available in the algorithm.*

2: *counter* = 0 — *user defined counter for stopping condition*

3: $\tau_{ij}(t) = c$; $\Delta\tau_{ij} = 0$ — *c is an initial value of pheromone value* τ_{ij} *given for every edge (i,j)*

4: $\eta_{ij} = 1$ — *visibility* η_{ij} *is initially given value 1 for all heuristics*

5: Set S_k and S_b as null — S_k *is each ant's solution,* S_b *is the best solution*

6: Place m ants on n heuristics

7: For $k = 1$ to m — *place the starting node of the ants.*
  Repeat until ants complete a journey
    For $k = 1$ to m do
      Select the next heuristic (node j) to apply based on pheromone and visibility
      Move the k-th ant to heuristic j
      Apply heuristic j to S_k to obtain new solution S'_k
      If $(S'_k < S_b)$
        $S_b = S'_k$
      Update visibility η_{ij} — *visibility is updated after a heuristic is applied*
      $t = t + 1$
    end for
    Update pheromone τ_{ij} — ***pheromone is updated after a tour is completed by ant k (local update)***
  counter ++

8: If (t == n) — *a tour for all ants is completed*
    If (any best solution is in this cycle)
      Update pheromone value τ_{ij} ***to best edge that produces best solution (global update)***
    counter = 0
    For all ants $S_k = S_b$
    Endif
  Endif

9: If ($t = t_{max}$)
    Output best solution S_b found
  Stop

Figure 4.6: Overview of ACO hyper-heuristic

4.9 Summary

In this chapter, two descriptions for the ant-based hyper-heuristic are presented. Firstly, a comparison between the ant system (Dorigo et al. 1996) and the proposed ant-based hyper-heuristic is listed. This section provides information on how the necessary input is represented. In the ant algorithm hyper-heuristic (Burke et al. 2003c and Chen et al. 2007), only edges that contribute to improvement to current solution are rewarded with pheromone values. However, in our approach, we are concerned with giving all the edges visited with some amount of pheromone, regardless the edges contributed to improvement or vice versa. The distribution of pheromone values is distributed proportioned to the performance done by the ants. The visibility value in the algorithm is dynamic throughout the algorithm. It is represented by the *cpu* time of each heuristic when it is applied. A heuristic is allowed to be visited as many times as possible compared to the original ant algorithm where a node is prohibited to be visited more than once.

The second algorithm, the ACO hyper-heuristic, uses the similar approach. It, however, differs in the way the pheromone values are updated. It introduces the global and local pheromone update. The global update uses the path that produces the best solution found at the current iteration to update the pheromone trail. This is to encourage the use of good-performing path and to increase the probability of choosing the specific path. The local update is performed after each ant performs a tour. This is to ensure that the pheromone values do not go unbounded. Both of these algorithms are used subsequently in the thesis. It is hoped that this chapter can be a reference for the experiments in the following chapters.

Chapter 5

Application to the Capacitated Vehicle Routing Problem

5.0 Introduction

In chapter 5, we have developed the ant-based hyper-heuristic. In this chapter, we will apply this approach to our first application problem; the capacitated vehicle routing problem (CVRP). The aim of this chapter is to demonstrate that our approach works well on the problem, using little domain knowledge of the problem and does not require extensive parameter tuning.

Ants are guided by pheromone trails and visibility values. Initially, we will conduct experiments to set the values of these parameters together with experiments to determine the starting positions for the ants and the evaporation rate values. The structure of this chapter is as follows; in section 5.1 we introduce the CVRP and its problem descriptions. Section 5.2 describes the low level heuristics that we utilize. Section 5.3 describes the experimental setup for the approach and section 5.4 presents the benchmark datasets that we use for the problem. In section 5.5, we present the solution method that we use to generate the initial solution. Section 5.6 describes experiments for establishing suitable parameters values for the algorithm. In section 5.7, we present the comparisons between our approach and previous ant algorithm hyper-heuristics. Section 5.8 presents the experiments to observe the effectiveness of our method. In order to identify the effects of these values, we have developed a random hyper-heuristic to serve as a

means of comparison with our framework. This is presented in section 5.9. The performance of ant-based hyper-heuristic is evaluated by comparing it with other methods. This is presented in section 5.10. Section 5.11 applies the ACO hyper-heuristic to the CVRP and finally, section 5.11 concludes the chapter.

5.1 Problem Formulation

The capacitated vehicle routing problem can be described as a fleet of vehicles p with capacity Q goods have to be delivered to customers $i \in N = \{1....n\}$ from a central depot $\{0\}$. The sum of demands on any route should not exceed the vehicle capacity Q. The objective of the CVRP is to minimize the total travel distance. The CVRP can be formulated as follows: a connected graph $G = (V,E)$ of $n+1$ nodes, with a set of customers with demand q_i is represented as $v_i \in V$, $i = 1.....n$ with v_0 as the depot. The mathematical formulation is as follows:

$$\bigcup_{p=1}^{v} R_P = N \qquad R_p \cap R_q = \emptyset, \forall_p \neq q \in V$$

$$\Sigma_{i \in R_p} d_i \leq Q \qquad \forall_p \in V$$

$$C(S) = \Sigma_{p \in V} C(R_p)$$

where:

v	number of vehicles
V	set of vehicles $V = (1....v)$
R_p	set of customers being served by vehicle p
$C(R_p)$	cost of individual tour length (route)
S	feasible solution of $S = \{R_1....R_p\}$
$C(S)$	total cost of each individual tour length $C(R_p)$

5.2 Low Level Heuristics

Low level heuristics are held in the nodes of the graph. Usually, they represent simple neighborhood structures such as swap, move or rules that were used by the user to construct the solutions. Initially, ants will be placed at a node on the search space (the directed graph). Referring to chapter 4, each ant will perform a tour by visiting a sequence of low level heuristics (nodes) by selecting a node after being guided by some information, the pheromone (evaluation function) and the visibility values (*cpu* time). Once an ant arrives at a particular node, it will apply the low level heuristic to its solution and generate a new solution. We have implemented 20 simple low level heuristics. Most of them are based on 2-opt moves. A 2-opt move deletes two edges, divides a tour into two parts, then reconnects the path in a number of ways. Other low level heuristics involve simple swap and moves which are based on route construction for VRP (Braysy and Gendreau 2005a). These moves will be performed if they satisfy the constraint.

The low level heuristics are described below:

1. ***CVRP_H1*** - choose a route at random and reverse a part of a tour between two randomly selected customers. For example, customer 2 and 7 are selected. The customers in between these customers will be reversed.

   Initial tour : 0 1 **2 3 4 5 6 7** 8 0

   Resulting tour : 0 1 **2 6 5 4 3 7** 8 0

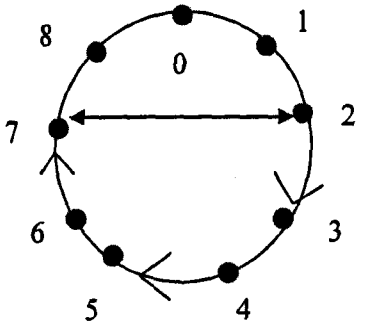

Figure 5.1(a): initial tour

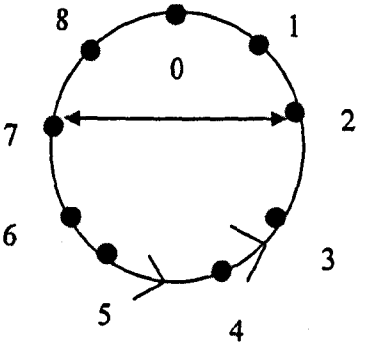

Figure 5.1(b): resulting tour

2. ***CVRP_H2*** - choose a route at random and perform 2-opt moves on the route. The procedure is done for every customer starting from the first customer after the depot. For example, edges between customer 2 and customer 3 and customer 6 and customer 7 are selected. 2-opt moves are done on the route.

   Initial tour : 0 1 **2 3** 4 5 **6 7** 8 0

   Resulting tour : 0 1 2 6 5 4 3 7 8 0

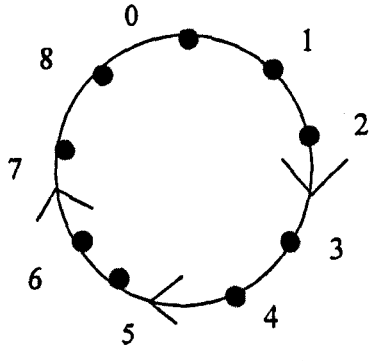

Figure 5.2(a): initial tour

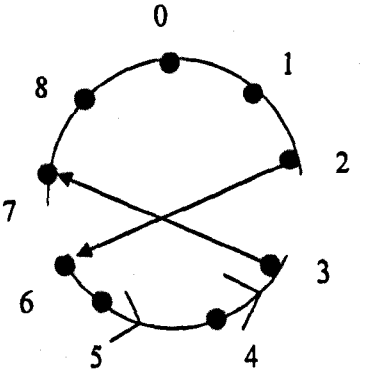

Figure 5.2(b): resulting tour

3. ***CVRP_H3*** - 2 opt procedure will be performed on all route. The procedure is demonstrated in the example above (no 2).

4. ***CVRP_H4*** - Swap 2 customers on a randomly selected route until no further improvement is made. For example, customer 2 and 7 are selected and will swap positions.

Initial tour : 0 1 **2** 3 4 5 6 **7** 8 0

Resulting tour : 0 1 **7** 3 4 5 6 **2** 8 0

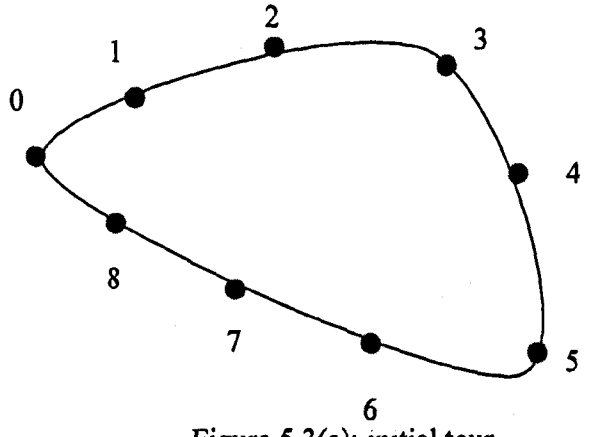

Figure 5.3(a): initial tour

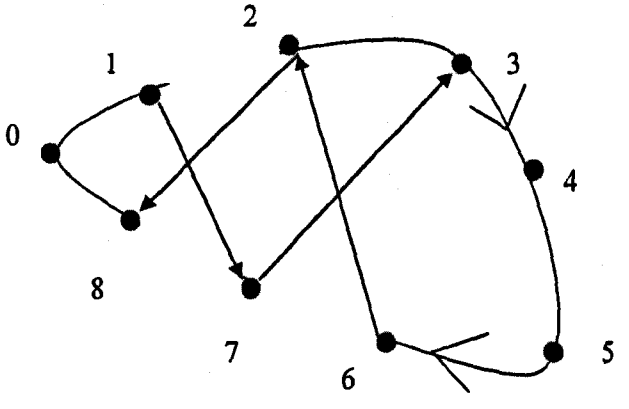

Figure 5.3(b): resulting tour

5. ***CVRP_H5*** - 2 routes will be selected at random. A customer will be chosen randomly on each route. We pick adjacent customers for the chosen customers on both routes. These adjacent customers will then be swapped. For example, customer 4 is chosen from the first route and customer 15 is chosen from route 2. Adjacent customer 5 will swap positions with customer 16 on the next route.

Initial tour: route 1 : 1 2 3 **4 5** 6 7 8 route 2 : 9 10 11 12 13 14 **15 16** 17

Resulting tour : route 1: 1 2 3 **4 16** 6 7 8 route 2 : 9 10 11 12 13 14 **15 5** 17

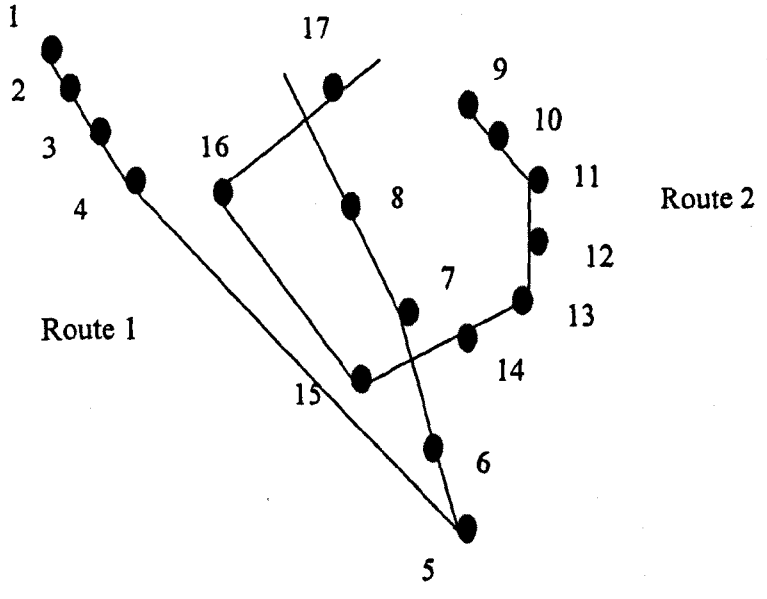

Figure 5.4(a): initial tour

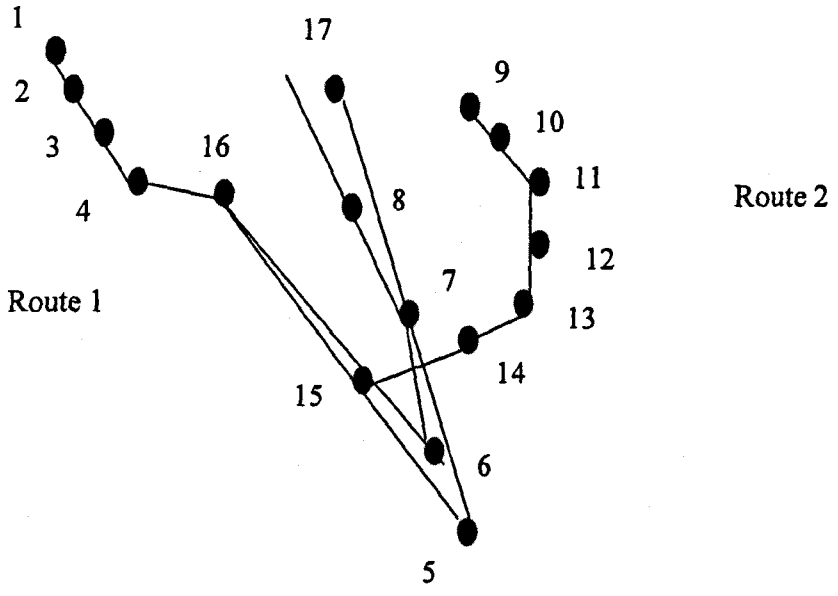

Figure 5.4(b): resulting tour

6. ***CVRP_H6*** - The same procedure as ***CVRP_H5*** is done on ***CVRP_H6*** but on two adjacent routes. (See example as above procedure (no 5)).

7. ***CVRP_H7*** - 2 routes will be selected at random. Swap end portion of a route with the first portion of another route. For example, customer 8 from route 1 and customer 9 from route 2 are selected and will swap positions.

   Initial tour: route 1 : 1 2 3 4 5 6 7 **8** route 2 : **9** 10 11 12 13 14 15 16 17

   Resulting tour : route 1: 1 2 3 4 5 6 7 **9** route 2 : **8** 10 11 12 13 14 15 16 17

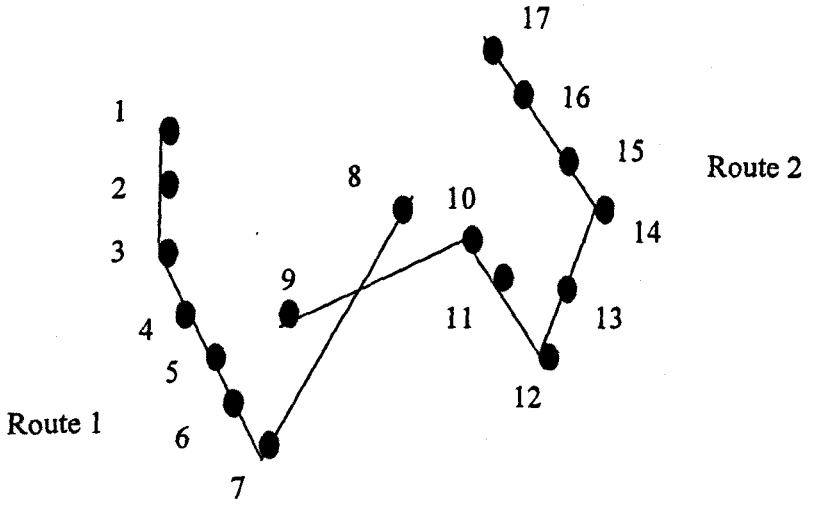

Figure 5.5(a): initial tour

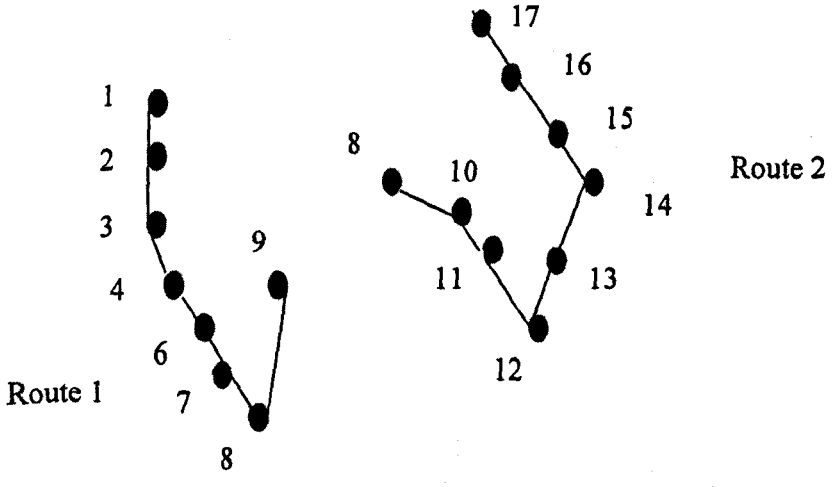

Figure 5.5(b): resulting tour

8. *CVRP_H8* - Move a customer from a random route to the nearest customer of another random route. In this procedure, we have developed a function to identify the nearest customer in another route to a selected customer. For example, customer 6 from route 1 will be moved to the front of customer 12 from route 2.

Initial tour: route 1 : 1 2 3 4 5 **6** 7 8 route 2 : 9 10 11 **12** 13 14 15 16 17

Resulting tour : route 1: 1 2 3 4 5 7 8 route 2 : 9 10 11 **6** **12** 13 14 15 16 17

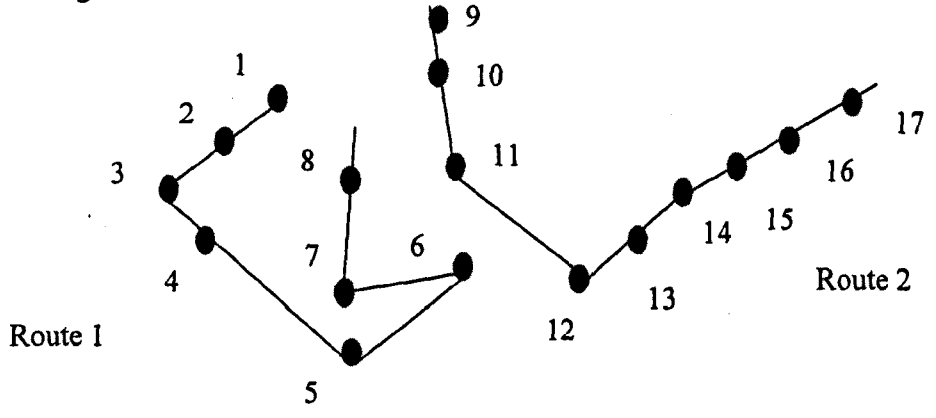

Figure 5.6(a): initial tour

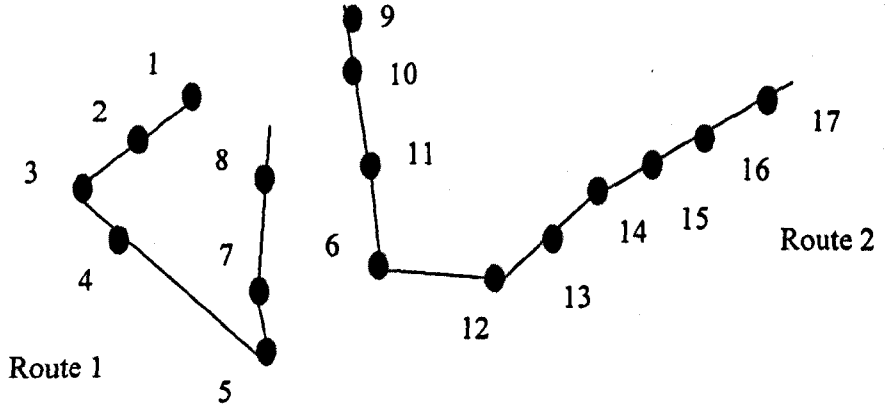

Figure 5.6(b): resulting tour

9. ***CVRP_H9*** - Choose 2 random routes. Move the last customer of the route to the first position of another route. For example, customer 8 from route 1 will be moved to the first position of route 2.

Initial tour: route 1 : 1 2 3 4 5 6 7 **8** route 2 : **9** 10 11 12 13 14 15 16 17

Resulting tour : route 1: 1 2 3 4 5 6 7 route 2 : **8 9** 10 11 12 13 14 15 16 17

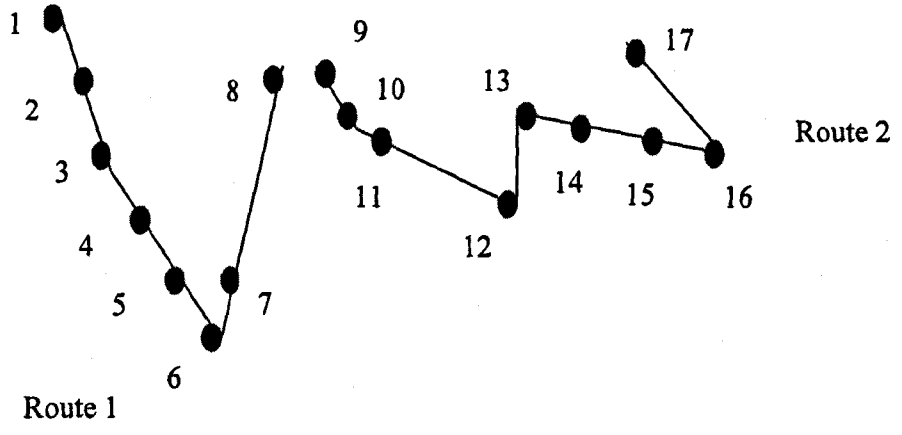

Figure 5.7(a): initial tour

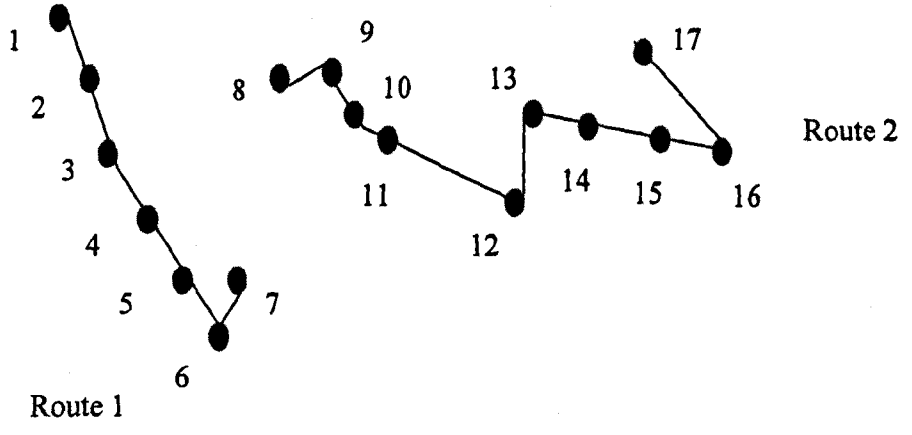

Figure 5.7(b): resulting tour

10. ***CVRP_H10*** - Choose 2 random routes and swap 2 customers (all) for the routes. For example, customers from route 1 will be interchangely swapped with all customers from route 2. In the example below, customer 6 will swap position with customer 13.

Initial tour: route 1 : 1 2 3 4 5 **6** 7 8 route 2 : 9 10 11 12 **13** 14 15 16 17

Resulting tour : route 1: 1 2 3 4 5 **13** 7 8 route 2 : 9 10 11 12 **6** 14 15 16 17

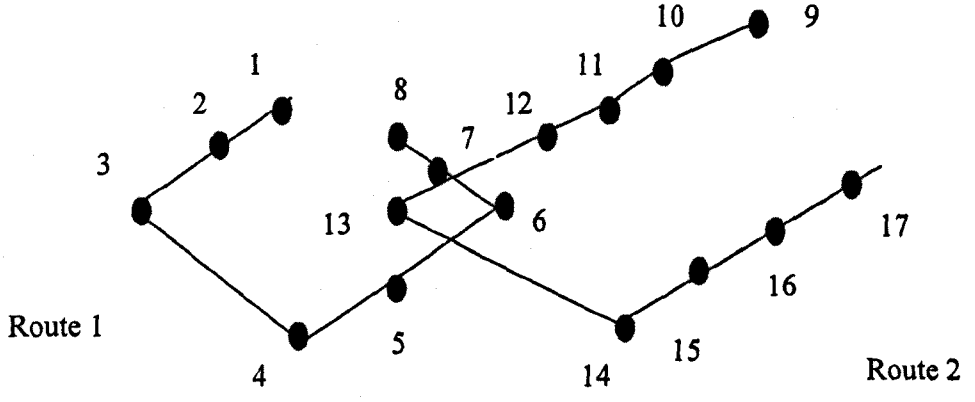

Figure 5.8(a): initial tour

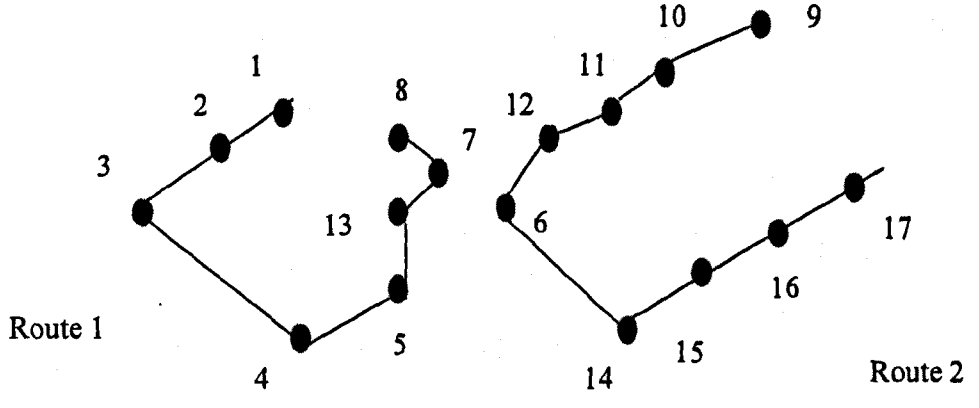

Figure 5.8(b): resulting tour

11. ***CVRP_H11*** - In this procedure, 2 adjacent routes will be chosen and 2 customers will swap position. An example of this procedure is the same as ***CVRP_H10*** except that it is performed on adjacent routes.

12. ***CVRP_H12*** - Choose 2 adjacent routes. The last customer of a random route will be moved to the first position of another random route. An example of this procedure is the same as ***CVRP_H9*** except that it is performed on adjacent route.

13. ***CVRP_H13*** - Choose 2 random routes and swap a customer from the route with the nearest customer of another route. An example of a diagram for this procedure is the same as ***CVRP_H10***. This, however, this is done for the nearest customer of another route.

14. ***CVRP_H14*** - Choose 2 random routes. Select a random customer and move a customer from a route after the nearest customer of another random route. For example, customer 5 is chosen to be moved after the nearest customer (customer 14) of another route.

Initial tour: route 1 : 1 2 3 4 5 **6** 7 8 route 2 : 9 10 11 **12** 13 14 15 16 17

Resulting tour : route 1: 1 2 3 4 5 7 8 route 2 : 9 10 11 **12 6** 13 14 15 16 17

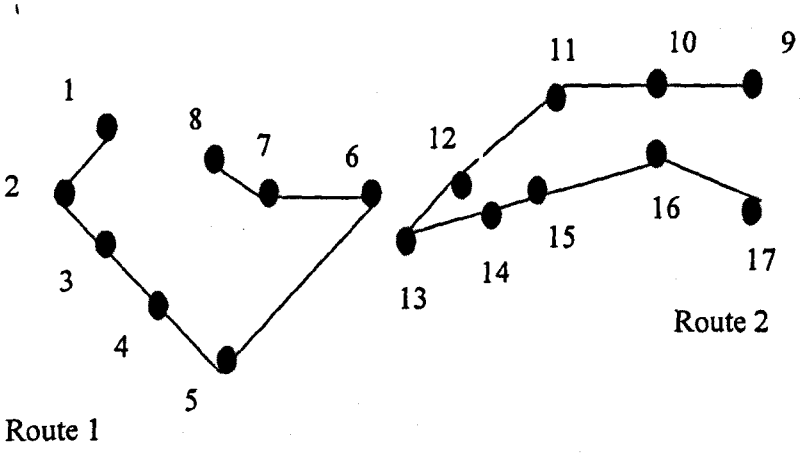

Figure 5.9(a): initial tour

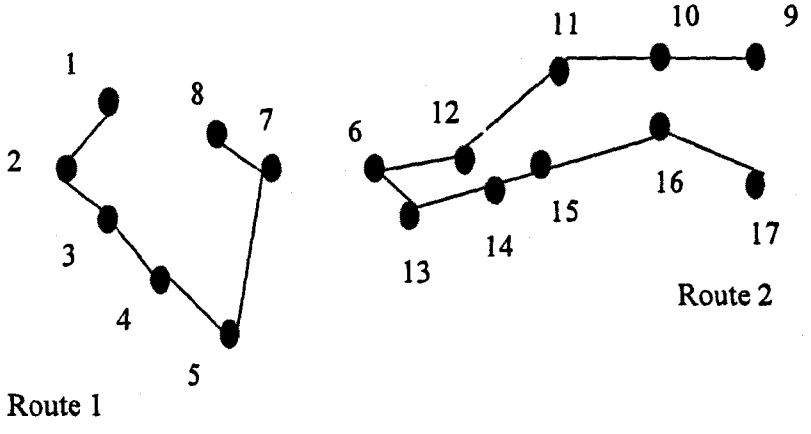

Figure 5.9(b): resulting tour

15. ***CVRP_H15*** - Choose a random route. Move a customer from a random route after the nearest customer of adjacent routes. An example of a diagram for this procedure is the same as ***CVRP_H14***.

16. ***CVRP_H16*** - Choose 2 routes at random. Move a customer from a route into another random position of another route. An example of a diagram for this procedure is the same as ***CVRP_H14***.

17. ***CVRP_H17*** - Move a customer from a random route into another random position of another adjacent route. For example, customer 6 is chosen from the first route and customer 13 is chosen from another random route. Customer 6 will be moved into the position of customer 13. An example of a diagram for this procedure is the same as ***CVRP_H14***.

18. ***CVRP_H18*** - 2 routes will be selected randomly. A customer will be chosen randomly on each route. We pick adjacent customers for the chosen customers on both routes. These chosen customers of route 1 will be swapped with adjacent customers of another route. An example of a diagram for this procedure is the same as ***CVRP_H10***.

19. ***CVRP_H19*** - The same procedure as ***CVRP_H18*** is done on ***CVRP_H19*** but on two adjacent routes. Refer to the example above; procedure (no 18).

20. ***CVRP_H20*** - 2 routes will be selected randomly. A customer will be chosen at random on each route. We pick adjacent customers for the chosen customers on both routes. Both customers from route 1 will be swapped with customers on route 2. For example, customer 4 is chosen from the first route and customer 10 is chosen from route 2. Their adjacent customers will be selected and all these customers will swap positions.

Initial tour: route 1 : 1 2 3 **4 5** 6 7 8 route 2 : 9 **10 11** 12 13 14 15 16 17

Resulting tour : route 1: 1 2 3 **10 11** 6 7 8 route 2 : 9 **4 5** 12 13 14 15 16 17

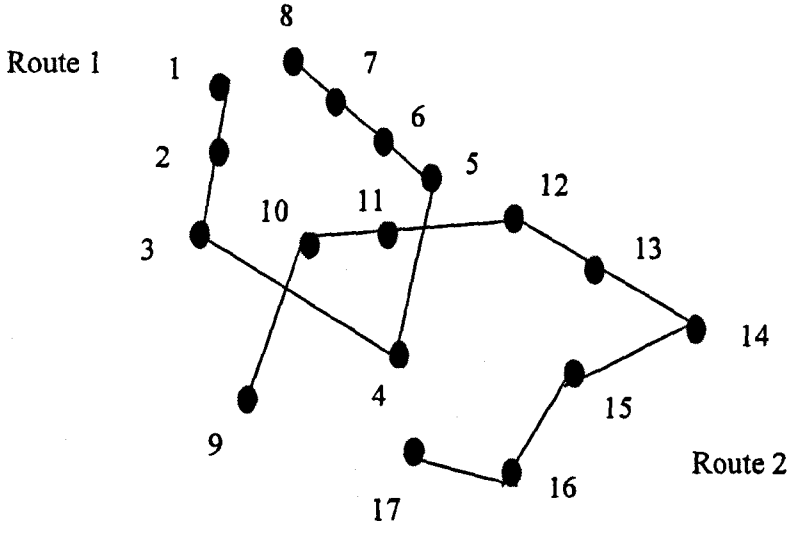

Figure 5.10(a): initial tour

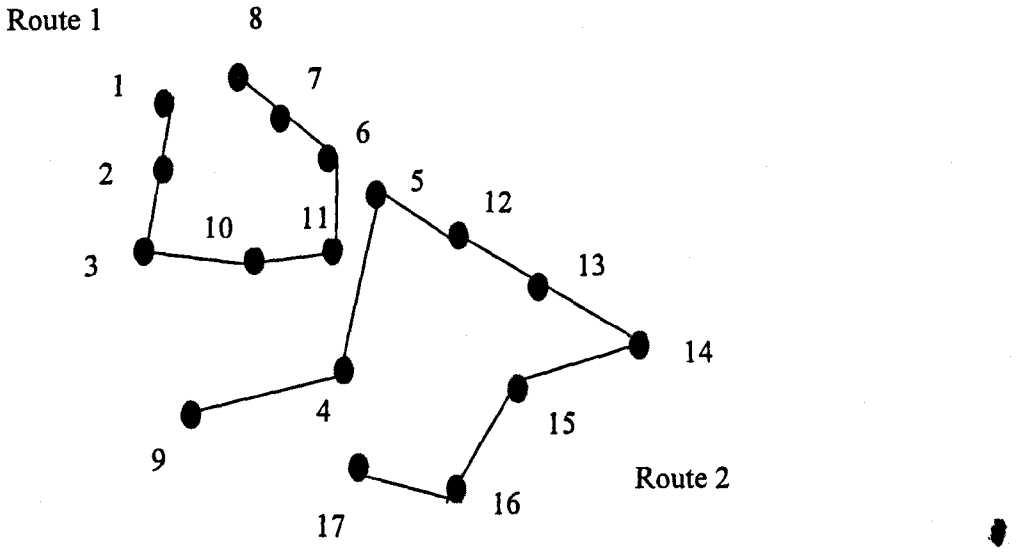

Figure 5.10(b): resulting tour

5.3 Experimental Setup

Our algorithm is coded in C++ using Microsoft Visual Studio 2008. All experiments are run on a PC Pentium R 3.4 GHz with 1 GB RAM running on Microsoft Windows 2000. Our ant-based hyper-heuristic are tested on a well known dataset which is described in the next section. We tested all experiments on 100 customers except for experiment to compare the results with random hyper-heuristic. We chose 100 customers as it is not a small and not a large dataset for the problem. In this work, all solutions generated are required to be feasible, that is all the low level heuristics will operate on a feasible search space. The results will be compared with the

best known or optimal solution reported in the literature. To reflect the generality of our approach we set the terminating conditions for this algorithm to 1000 iterations. However, if the algorithm does not meet any improvement for 100 iterations, it will be terminated and the best result will be returned. Among the objectives of experiments conducted in this chapter are:

- To demonstrate a relationship between the high and low level domain of hyper-heuristic.
- To establish appropriate parameter settings, both for the pheromone and visibility. It also identifies the appropriate value for evaporation for the pheromone values.
- To determine a suitable starting position for the artificial ants.
- To investigate the performance of the algorithm with different sets of low level heuristics.
- To investigate whether the artificial ants follows the path guided by the pheromone and visibility values or whether the artificial ants select paths randomly.
- To demonstrate that the algorithm can effectively select low level heuristics at each decision point.
- To demonstrate that the algorithm can produce competitive results when compared to other methods.

The null hypotheses used in the experiments are listed below:

- ***H1***: the three starting position are not significantly different from each other. Experiments are presented in section 5.6.1.
- ***H2***: Different numbers of low level heuristics are not significantly different from each other. Experiment is presented in section 5.8.2.
- ***H3***: The results for ant-based hyper-heuristic are not significantly different from random hyper-heuristic. The experiment is done in section 5.9.

5.4 The Benchmark Datasets

The algorithm is tested on 7 problems from Vehicle Routing Datasets (2003). Problem sizes ranging from 50 – 199 customers in this dataset are well known problems that are benchmark comparison for CVRP. The data with customer locations are described by coordinates. The first node in this dataset is the depot. The dataset consist of two classes of problem; randomly distributed problems and clustered problems. The number their classes of customers, the capacity of each instance and their optimal solutions are shown in Table 5.1.

Problem	Size	Type	Capacity	Vehicles	Optimal Solution
E-n51-k5	50	random	160	5	524.61
E-n76-k10	75	random	140	10	835.26
E-n101-k8	100(a)	random	200	8	826.14
M-n151-k12	150	random	200	12	1028.42
M-n200-k17	199	random	200	17	1291.5
M-n101-k10	100(b)	clustered	200	10	819.56
M-n121-k7	120	clustered	200	7	1042.11

Table 5.1: The description and optimal solutions for Christofides et al. (1969) (Vehicle Routing Datasets 2003)

5.5 Initial Solutions

The initial solutions for the CVRP have been generated using the sweep algorithm (Gillet and Miller 1974). The algorithm is described in chapter 3 (section 3.2.1.1.2).

5.6 Experiments for Determining Parameter Values

Ants apply probability transition rules (refer chapter 4.6) to construct a tour. In order to start a tour, there are certain characteristics an ant has to adopt:

i. a starting heuristic (node) where the ants will be placed.

ii. to construct a tour, ants are guided by the pheromone and heuristic values. α represents the influence of pheromone trail. The β represents the influence of visibility value.

iii. To reduce the influence of pheromone values on an edge, there is a need for evaporation procedures. This process is required so that the pheromone values do not grow unbounded. Furthermore, it helps in reducing the influence of pheromone in the early stages of search, where ants typically build poor solutions.

We conduct experiments to determine appropriate parameter setting for our approach. In Burke et al. (2005b), the number of ants was set to be equal to the number of nodes in the network. However, in our work, we set the number of ants to 10 (except for the experiment in section 5.6.1, we use 20 ants) as we assume if we set the ants to be equivalent to the number of nodes (heuristics), the method will consume more computational time. Too few ants will limit the exploration of the search space; therefore, we set the number of ant to 10 ants as we consider it is not too many and not too few to explore the search space of heuristics.

5.6.1 Experiments with Different Starting Positions

In this experiment, the aim is to analyse the influence of starting positions of the ants. As part of the investigation, we use 20 ants and place the ants at three distinct places:

i. all ants are placed on the first heuristic (*CVRP_H1*)

ii. randomly placed on any heuristics

iii. place one ant on each heuristic.

This experiment was done on dataset with 100 customers (E-n101-k8). Each different starting position was run for 1000 iterations. However, if it does not give any improvement for 100 iterations, the algorithm will terminate. The best, worst, average solution and the standard deviation for 30 runs are recorded. The results for each placement are presented in Table 5.2.

Experiment	Starting position	Best	Worst	Average	Std Dev
1	Random	**842.34**	879.60	866.96	9.84
2	Heuristic 1	857.91	965.80	883.55	21.57
3	Each heuristic	854.95	916.00	882.82	15.35

Table 5.2: Results for different starting positions

As shown in this table, placing the ants randomly on heuristics available produced good solutions compared to other starting positions. To evaluate the null hypotheses *H1* (see section 5.3), a t-test with 95% confidence level is performed and the results are compared and presented in Table 5.3. Here, the comparison is performed between the various placing to observe any effects of the initial placement for the ants. The data used for these tests are shown in Appendix A.

Experiments	Random	Heuristic 1	Each heuristic
Random	–	reject	reject
Heuristic 1	–	–	accept
Each heuristic	–	–	–

Table 5.3: Results of t-test for different starting positions

As shown in table 5.3, null hypotheses *H1* is rejected for experiment 1. This shows that by placing the ants randomly as starting position is significantly different compared to the other two placements which indicates that it has an influence of guiding the ants to have better explorations on the search space. Placing all ants on the first heuristic or placing the ants individually on each heuristic does not show to be any statistically different from each other (null hypotheses *H1* is accepted). This implies that placing the ants on these two positions has produced results that are not different from each other. Therefore, we chose to place the ants randomly subsequent for all experiments.

5.6.2 Experiments with Different Pheromone and Visibility Rates

Ants communicate with each other via pheromone trails. While travelling from one node to another, ants leave some pheromone trails on the path they travel. Ants sense the pheromone and choose probabilistic path with strong pheromone trails. Visibility is heuristic information that helps the ants build good quality tours. For example, in the TSP, the visibility value is represented by inversely proportioning the distance between city i and city j. We perform parameter test to obtain suitable setting for pheromone trails and visibility values. Basically in an ant algorithm, shorter paths receive more pheromone trails left by the ants. The α value represents the influence of pheromone trails. The β value represents the influence of visibility value. If $\alpha = 0$, only heuristics with low computation time will be selected thus restricting the search space to be explored more thoroughly. If $\beta = 0$, the only path with strong trail of pheromone will be selected. This will lead all ants to follow the same path, hence resulting in quick convergence.

In this experiment, all ants will be placed randomly on any heuristics. Higher improvements on the objective function will reward higher pheromone trail on the path travelled. Visibility in this algorithm is presented by the *cpu* time taken by each ant to complete its task. The lower the *cpu* time taken, the higher the visibility value given to the node. Each combination of pheromone and visibility values is run for 10 times for each of the 1000 iterations. The best, worst, average solutions and the standard deviation are recorded.

α	β	**best**	**worst**	**average**	**Std dev**
0.1	0.9	860.29	888.14	871.26	8.16
0.2	0.8	861.58	873.71	866.97	4.09
0.3	0.7	849.19	892.02	871.64	7.93
0.4	0.6	858.85	908.71	876.30	10.87
0.5	0.5	**842.34**	879.60	872.30	10.87
0.6	0.4	864.95	888.30	872.99	6.84
0.7	0.3	859.13	878.88	869.03	3.73
0.8	0.2	855.13	885.61	871.89	9.37
0.9	0.1	859.55	897.53	874.00	8.60

Table 5.4: Results for different pheromone and visibility rate

In can be seen in Table 5.4 that the combinations of $\alpha = 0.5$ and $\beta = 0.5$ have produced good solution compared to other combinations; therefore, we chose these values to present the pheromone trail and visibility values throughout all experiments.

5.6.3 Experiments with Different Evaporation Rates

Pheromone evaporation is an important activity to determine the level of trails existed on a particular path. We perform parameter tests to identify the best setting for the evaporation rate of the pheromone values. Each evaporation rate is run for 10 times for 1000 iterations each. The best, worst, average solutions and the standard deviation are recorded.

Evaporation rate	best	worst	average	Std dev
0.1	846.49	880.74	862.51	12.80
0.2	850.27	884.80	867.46	10.10
0.3	**843.63**	**876.86**	**868.07**	**9.40**
0.4	859.75	881.51	867.60	8.00
0.5	847.14	879.55	863.40	8.00
0.6	862.82	913.80	877.23	17.30
0.7	866.85	891.42	877.12	8.70
0.8	862.54	886.57	873.81	7.20
0.9	851.31	880.57	869.95	9.80

Table 5.5: Results for different evaporation rate

Results in Table 5.5 show that evaporation rate 0.3 is the best setting to obtain best solutions among other rates. We will use 0.3 as evaporation rate throughout all other experiments.

5.7 Comparisons of Ant Hyper-heuristics

The objective of this section is to compare previous approaches in Burke et al. (2003c) and Chen et al. (2007) to our approach. We utilise the same parameter values for all approaches ($\alpha = 0.5$, $\beta = 0.5$ and evaporation rate is 0.3). The experiment was conducted on 50 customers (E-n51-k5) with 30 runs. The results are presented in the following Table 5.6. We can see that our ant-based hyper-heuristic has outperformed both the ant algorithms hyper-heuristic in terms of best known results, average and standard deviation. Chen et al. (2007) performed better than Burke et al. (2003c). The time taken by our approach is also better than the two approaches. This shows that the performance of ant hyper-heuristics can be improved by introducing new pheromone updates where the distribution of pheromone values is deposited proportioned to their quality of solutions to all visited edges. The visibility values measured are the computational time taken for a heuristic to complete its task.

	Chen et. al	time	Burke et al.	time	Ant-based hh	time
	549.341	85.6	687.81	33.6	539.681	3.2
	546.552	79.0	712.455	31.5	532.996	10.7
	546.383	110.5	714.408	22.0	542.639	4.8
	543.594	105.2	758.814	25.8	539.681	6.6
	543.594	118.4	591.864	62.7	542.47	4.8
	552.089	64.8	692.996	31.9	543.594	3.2
	546.552	91.1	777.376	22.9	543.594	6.5
	551.257	95.8	599.312	18.0	540.224	4.8
	543.594	114.1	765.838	18.8	543.594	3.2
	543.594	72.1	679.081	46.0	541.864	1.5
	543.594	75.2	714.071	31.5	532.996	7.3
	545.971	87.1	572.95	66.6	542.639	3.2
	546.552	107.7	785.334	17.2	543.594	4.9
	543.594	131.8	547.809	170.1	542.639	3.2
	546.552	81.9	702.383	11.7	539.681	3.2
	543.594	67.5	543.013	71.3	542.47	3.2
	547.809	70.1	693.441	30.9	542.639	4.9
	543.594	100.4	689.459	32.5	539.681	3.2
	559.224	47.1	546.383	68.3	543.594	6.5
	543.594	97.4	572.722	232.3	532.996	20.6
	543.013	65.3	756.822	11.3	539.681	4.9
	547.809	106.5	672.183	25.0	539.681	4.9
	556.106	102.0	769.029	12.8	543.594	3.2
	546.383	105.4	720.868	16.6	543.594	3.2
	543.594	76.7	751.013	26.1	543.594	6.5
	546.383	82.1	584.393	45.6	532.996	5.8
	559.224	65.1	740.478	34.0	548.209	3.1
	543.594	98.4	666.405	16.6	543.594	3.2
	543.594	87.7	547.809	75.6	539.681	3.3
	547.809	93.5	599.068	183.6	539.681	4.9
best	543.013	47.13	543.013	11.34	532.996	1.54
worst	559.224	131.84	785.334	232.30	548.209	20.57
average	546.94	89.52	671.85	49.76	540.919	5.08
std dev	4.52	19.15	79.98	53.38	3.71	3.45

Table 5.6: Comparisons of three different ant hyper-heuristics

5.8 Effectiveness of the Ant-Based Hyper-heuristic

In this section, we conduct experiments to determine the effectiveness of our framework. We first perform an experiment to determine the frequency at which the low level heuristics are applied by the hyper-heuristic. Furthermore, the outcome of this experiment will determine our selection of how many low level heuristics are required to produce good quality solutions in the next experiment. We then introduce random hyper-heuristic to serve as a comparison for our ant-based hyper-heuristic. In this random hyper-heuristic, all heuristics (nodes) will have the same probability to be selected by the ants. In the experiments, we will investigate whether the pheromone trails and visibility play some role in guiding the ants to select good heuristics. Lastly, we present our results and compare them with other approaches available in the literature.

5.8.1 Heuristic Calls

In this section, we perform experiments to determine which heuristics are frequently being applied and also heuristics that are less frequently applied. At the beginning of the search, a set of low level heuristics is selected to be applied to the initial solution which is the current solution at the time. The algorithm starts with a feasible solution. Each ant will select a series of low level heuristics. The total number of low level heuristics applied by an ant is considered as a tour. The new solution obtained after employing the low level heuristics will replace the previous solution and it is the best solution among the current solution available. The process is repeated for 1000 iterations and will stop when it meets the termination criterion or if it does not give any improvement for 100 iterations. Table 5.7 presents the results together with the average and the

standard deviation and plots a frequency chart to observe the behaviour of these heuristic calls. Figure 5.11 demonstrates the frequency chart of the low level heuristics for dataset with 100 customers. In Figure 5.11, the x-axis represents the frequency calls of the low level heuristics and y-axis represents the low level heuristics being employed throughout the search. We observe that ***CVRP_H1*** has been called the most frequent. ***CVRP_H7*** is the second frequent being called. Other frequent heuristic that are being called are ***CVRP_H2***, ***CVRP_H5***, ***CVRP_H9***, ***CVRP_H10***, ***CVRP_H11***. The least call of this low level heuristic is heuristic 20. Other least called heuristics are ***CVRP_H16*** and ***CVRP_H17.*** We will further adopt these findings in the following section.

run	h1	h2	h3	h4	h5	h6	h7	h8	h9	h10
1	3036	2826	2878	2832	2903	2880	2963	2781	2840	2808
2	2977	2831	2836	2817	2804	2815	2962	2969	2812	2853
3	2933	2909	2791	2891	2850	2864	2900	2775	2841	2883
4	2973	2839	2826	2821	2854	2815	2869	2928	2880	2796
5	2973	2861	2980	2854	2816	2812	2858	2790	2878	2876
6	2942	2924	2835	2813	2983	2958	2845	2831	2914	2867
7	2902	2909	2759	2829	2846	2824	2942	2850	2891	2899
8	2970	2833	2937	2799	2780	2828	2866	2882	2827	2871
9	2977	2951	2775	2813	2886	2857	2924	2909	2927	2905
10	2945	2889	2918	2923	2925	2915	2803	2793	2874	2846
average	2963	2877	2854	2839	2865	2857	2893	2851	2868	2860.4
std dev	35.52	45.01	73.13	39.29	60.98	49.02	53.75	68.59	37.62	35.81

Table 5.7: The frequency calls of low level heuristics for 100 customers

h11	h12	h13	h14	h15	h16	h17	h18	h19	h20
2887	2784	2769	2808	2871	2761	2842	2778	2829	2924
2879	2858	2836	2885	2806	2820	2797	2899	2799	2828
2819	2827	2870	2791	2857	2848	2765	2840	2724	2735
2954	2902	2797	2814	2771	2757	2806	2795	2812	2761
2872.3	2845.5	2826	2836	2830	2805	2814	2831	2821	2793
39.76	46.56	31.00	52.73	65.76	45.19	62.74	58.76	44.20	58.66

Table 5.7: The frequency calls of low level heuristics for 100 customers (continued)

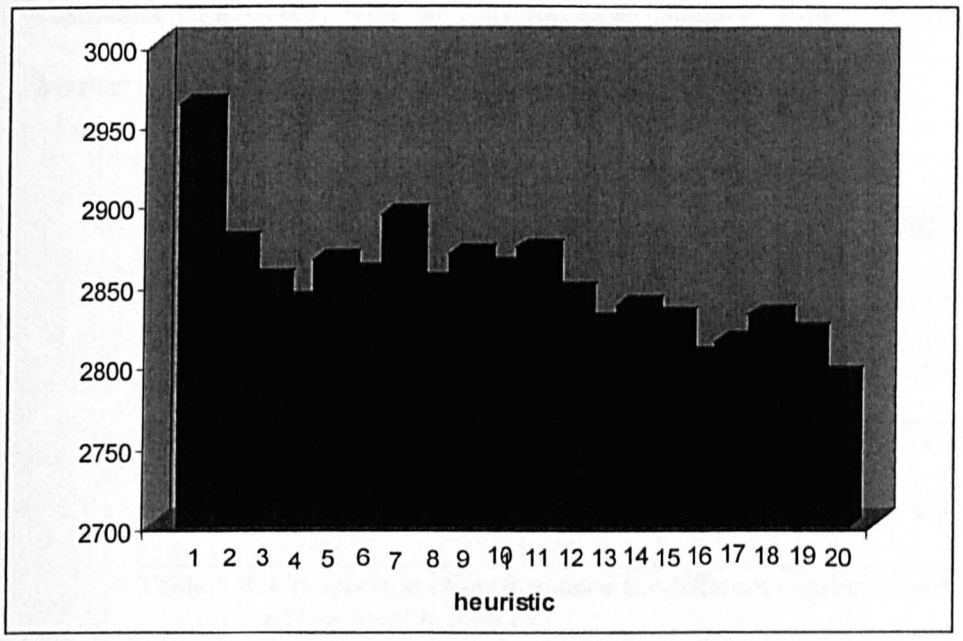

Figure 5.11: The frequency chart of the low level heuristics

5.8.2 Experiments with Different Sets of Low Level Heuristics.

We further investigate the best number of low level heuristics required to produce good solutions. We test the algorithm with four different categories of heuristics based on their performance. In order to identify these categories, an experiment to capture the calls of every heuristic is done in the above section (section 5.8.1). Set A consists of ten most frequent heuristics being called (heuristic 1,2,3,5,6,7,8,9,10,11). Set B consists of ten heuristics that are less being called during the procedure (heuristic 4,12,13,14,15,16,17,18,19,20). Set C consists of ten heuristics that are chosen intuitively which we predict to perform well in exploring the search

space (heuristic 1,2,3,4,5,10,13,14,16,19). Set D is the combination of the twenty heuristics available. We include heuristics in set B (bad performing heuristics) as well as to encourage the search to explore other promising areas of search space. The experiment was conducted on 100 customers (E-n101-k8) with 30 runs for each category. Table 5.8 presents the best, worst, average results and the standard deviation for each experiment.

Heuristics	Best	Worst	Average	Std Dev
Set A	863.96	916.24	884.03	12.64
Set B	879.83	986.11	920.31	25.63
Set C	**838.65**	889.29	871.10	11.20
Set D	842.34	879.60	863.60	9.90

Table 5.8: Comparison of performance for different combination number of low level heuristics

The results demonstrate that the algorithm perform best when we use ten intuitive heuristics (set C). However, we observe that overall twenty combinations (set D) of all low level heuristics have better solutions even though their best solution is slightly worse than ten intuitive heuristics. The worst, average results and standard deviation of combination of these low level heuristics are better than set C heuristics. To further assess null hypotheses *H2* (describe in section 5.3), statistical test (t-test) with 95% confidence level is conducted for each experiment. The results of these statistical tests are summarised in Table 5.9. The data used for these tests are shown in Appendix D.

Experiments	Set A	Set B	Set C	Set D
Set A	X	reject *H2*	reject *H2*	reject *H2*
Set B	X	X	reject *H2*	reject *H2*
Set C	X	X	X	reject *H2*
Set D	X	X	X	X

Table 5.9: T-test for different combination number of low level heuristics

As shown, for all experiments, null hypotheses *H2* are rejected indicating that all these combinations are statistically different to each other. This implies that varying the number of low level heuristics has an influence in generating good solutions. Finally, we conclude that higher number of heuristics used in this approach will guide the ants to better solutions.

5.9 Random Hyper-heuristic

We develop random hyper-heuristic to serve as a comparison for our ant-based hyper-heuristic. In this random hyper-heuristic, all heuristics (nodes) will have the same probability to be selected by the ants. Our aim in the experiment is to investigate whether the pheromone trails and visibility play some roles in guiding the ants to select good heuristics. We apply the random hyper-heuristic on all datasets. The experiment is run for 30 runs. Table 5.10 presents the best, worst; average results and the standard deviation for the random hyper-heuristic and we make comparisons with our approach. From the table, for all tested datasets, it can be seen that ant-

based hyper-heuristic clearly outperformed the random hyper-heuristic in the best solutions found. However, in terms of average and standard deviation; out of seven datasets tested, results show that ant-based hyper-heuristic has outperformed the random hyper-heuristic in five datasets. This probably shows that ant-based hyper-heuristic shows better consistency than random hyper-heuristic. To further investigate the similarity of these algorithms, we apply a statistical test (t-test) to access the null hypotheses *H3* (as described in section 5.3). The null hypotheses *H3* for this experiment are that the results of the ant-based hyper-heuristic algorithm are not significantly different from the random hyper-heuristic. The results of this statistical test with 95% confidence interval are presented in Table 5.11. The data used for these tests are shown in Appendix E.

As shown for most of the experiments, null hypotheses *H3* are rejected, except in experiment 4 for dataset 100(b). It is demonstrated that these algorithms have no similarity. However, for dataset with 100(b) customers, it is found that the ant-based hyper-heuristic does not have significant difference with the random hyper-heuristic. Datasets with 100(b) customers are clustered problems which with initial value generated with sweep algorithm do not perform too well. This is in line with our expectation. This indicates that the ant-based hyper-heuristic does not randomly pick low level heuristics; instead we can conclude that it is guided by the pheromone and the visibility values.

Dataset	**Anthh** *best*	**Anthh** *ave*	**Anthh** *worst*	**Anthh** *Std dev*	**Rndhh** *best*	**Rndhh** *ave*	**Rndhh** *worst*	**Rndhh** *Std Dev*
50	**532.99**	540.92	548.21	3.71	542.82	554.66	584.59	12.07
75	**863.84**	875.02	889.58	6.24	870.34	907.90	949.80	23.35
100(a)	**842.34**	863.60	879.60	9.90	864.81	886.94	923.81	17.13
100(b)	**910.85**	963.84	988.13	22.61	911.65	961.75	987.31	21.24
120	**1107.77**	1224.65	1293.39	49.91	1140.54	1202.51	1260.60	32.57
150	**1076.21**	1097.55	1115.04	9.54	1084.77	1103.75	1155.46	14.73
199	**1371.11**	1397.06	1436.76	14.57	1387.98	1419.97	1491.16	21.23

Table 5.10: Comparisons in the performance of ant-based hyper-heuristic (Anthh) and random hyper-heuristic (Rndhh)

anthh / rndhh	50	75	100(a)	100(b)	120	150	199
50	reject *H3*	X	X	X	X	X	X
75	X	reject *H3*	X	X	X	X	X
100(a)	X	X	reject *H3*	X	X	X	X
100(b)	X	X	X	*accept H3*	X	X	X
120	X	X	X	X	reject *H3*	X	X
150	X	X	X	X	X	reject *H3*	X
199	X	X	X	X	X	X	reject *H3*

Table 5.11: T-test for comparing the performance of ant-based hyper-heuristic (anthh) and random hyper-heuristic (rndhh)

5.10 Comparisons with Other Methods

In this section, we compare our ant-based hyper-heuristic to other metaheuristics such as tabu search, simulated annealing, genetic algorithm and the ant algorithm. Our results and deviations to best known together with computations times are shown in Table 5.12. In the table, the best known result for the problem is presented in bold. Our results do not produce any better results. For some datasets, however, we are able to produce better results than some other methods. Our approach performs better than the ant system for datasets with 75, 100(a) (problem 3), 150 and 199 customers. For datasets with 120 and 199 customers, we produce better results than the simulated annealing algorithm. For random distributed problems (problems: 1,2,3,6,7), we are able to produce solutions that have average deviation less than 6% from the best known solutions

while for clustered problems (problem: 4,5), our approach only manage to produce solutions with average deviation less than 11%. We conclude that even though our approach does not produce the best known results, we have managed to produce good results by only using only simple low level heuristics (simple 2-opt procedure and simple move and swap procedure). Furthermore, our approach does not involve extensive parameter tuning. In the beginning of the approach, we have done a parameter setting and have used it throughout the run of every dataset.

Problem	Dataset	Best known	Taillard	*Time*	Tabu-route	*Time*	Simulated Anealing	*Time*
1	51	**524.61**	**524.61**	*1.12*	**524.61**	*6.0*	528	*2.79*
2	76	**835.26**	**835.26**	*1.18*	835.32	*53.8*	838.62	*107.2*
3	100(a)	**826.14**	**826.14**	*11.25*	**826.14**	*18.4*	829.18	*155.6*
4	100(b)	**819.56**	**819.56**	*6.79*	**819.56**	*16*	826	*10.53*
5	120	**1042.11**	**1042.11**	*23.31*	**1042.11**	*22.2*	1176	*5.26*
6	150	**1028.29**	1028.42	*51.25*	1031.07	*58.8*	1058	*83.54*
7	199	**1291.45**	1298.79	*32.88*	1311.35	*90.9*	1378	*38.64*

Table 5.12: Comparisons of ant-based hyper-heuristic to other methods

Problem	Genetic Algo	*Time*	Ant system	*Time*	I.ant system	*Time*	Rndhh	*Time*	Anthh	*Time*
1	**524.61**	*<1*	**524.61**	*0.6*	**524.61**	*0.1*	542.82	*3.57*	532.99	*7.28*
2	**835.26**	*3.06*	870.58	*2.4*	844.31	*1.3*	870.34	*10.56*	863.84	*6.13*
3	**826.14**	*8.28*	879.43	*11.3*	832.32	*3.8*	864.81	*49.26*	842.34	*17.74*
4	**819.56**	-	819.96	*10.1*	**819.56**	*5.0*	911.65	*32.26*	910.85	*16.09*
5	**1042.11**	*35.72*	1072.45	*16.2*	1065.21	*9.2*	1140.54	*29.43*	1107.77	*30.43*
6	1030.46	*32.58*	1147.41	*28.5*	1061.55	*18.4*	1084.77	*45.75*	1076.21	*17.74*
7	1296.39	*56.07*	1473.4	*82.2*	1343.46	*87.6*	1387.98	*138.69*	1371.11	*54.39*

Table 5.12: Comparisons of ant-based hyper-heuristic to other methods (continued)

5.11 ACO Hyper-heuristic Applied to CVRP

Finally we applied the ACO hyper-heuristic to the CVRP. The aim of this section is to evaluate the effect of the global and local updating rules for the pheromone values. We compare the results to the ant-based hyper-heuristic and present both results in Table 5.13. From the findings, we observe that in most datasets, ant-based hyper-heuristic has outperformed the ACO hyper-heuristic. However, for datasets 100(b) and 150 customers, ACO hyper-heuristic has outperformed the ant-based hyper-heuristic. In terms of average results, ant-based hyper-heuristic performed better than the ACO hyper-heuristic in four datasets (datasets 50, 75, 100(a), 120). For standard deviation, however, ACO hyper-heuristic performs better in four datasets (datasets 50, 120, 150, 199). ACO hyper-heuristic has obtained the worst solution in five datasets (datasets 50, 75,100(a), 100(b)). These observations indicate that ant-based hyper-heuristic performs better than ACO hyper-heuristic. It appears that ACO hyper-heuristic does not take into account the global and local updating procedure. This is inconsistent with the performance of ACO algorithm in the literature where by modifying the process of updating the pheromone values, the results are better (Dorigo and Di Caro 1999c). This is probably because of the difference in the search space between the metaheuristics and hyper-heuristic. Metaheuristics operate in the search space of solutions however hyper-heuristic operate in the search space of heuristics. Heuristics perform differently at each decision point, thus there is no guarantee that the same heuristic will perform the same or better for the next iteration. Therefore, it can be seen that by enforcing global and local updates of pheromone values does not appear to contribute to any success in obtaining better results than ant-based hyper-heuristic

Datasets	*Best* Anthh	*Average* Anthh	*Worst* Anthh	*Std Dev* Anthh	*Best* ACOhh	*Average* ACOhh	*Worst* ACOhh	*Std Dev* ACOhh
50	**532.99**	540.92	548.21	3.71	539.7	543.6	548.5	1.9
75	**863.84**	875.02	889.58	6.24	868.9	885.7	900.0	7.4
100(a)	**842.34**	863.6	879.6	9.9	854.6	876.6	906.5	11.6
100(b)	910.85	963.84	988.13	22.61	**851.1**	911.7	988.5	33.4
120	**1107.77**	1224.65	1293.39	49.91	1116.0	1226.8	1297.1	47.6
150	1076.21	1097.55	1115.04	9.54	**1068.9**	1080.0	1088.6	4.5
199	**1371.11**	1397.06	1436.76	14.57	1377.9	1393.5	1410.0	8.3

Table 5.13: Results of ACO hyper-heuristic and ant-based hyper-heuristic

5.12 Summary

In this chapter, we applied our proposed approach to solve the CVRP. Although results presented in this paper do not produce any better results than in the literature, we have proposed a new algorithm that tries to find a general approach to solve the CVRP. In this work, we present an ant-based hyper-heuristic as a high level selector to pick and combine several low level heuristics. One major advantage of this work is we use simple swap and move, knowledge of poor low level heuristics that do not require expertise in the problem domain and the same parameter for all problem instances. In addition, we have done some experimental work to observe the effects of pheromone and visibility values in guiding the ants to explore the search space. We compare the technique with random hyper-heuristic where all heuristics will have the same probability to be selected. Statistical tests show that these two techniques are statistically significant, except for dataset 100(b) customers, where there is no statistical significance between these two techniques. The dataset with 100(b) customers represents clustered problems which with initial value generated with sweep algorithm does not perform too well.

Further research on investigating the number of heuristics needed to produce good solutions is done. Results show the number of heuristics is important to find good solutions for CVRP. It indicates that the larger the number of low level heuristics used, the better the average and the standard deviation. We investigate the performance of ACO hyper-heuristic compared to ant-based hyper-heuristic and the findings show that ant-based hyper-heuristic perform better in most datasets. Ant-based hyper-heuristic updates pheromone values on all visited edges. The

distribution of pheromone is proportioned to the quality of improvement performed by the ants. In contrast to the ACO hyper-heuristic, the global updates of pheromone intend to update pheromone on the best performing edge and local pheromone update is performed after each ant performs a tour. A hyper-heuristic operates in the search space of heuristics. At each decision point, the heuristic performs differently. Bad tours become highly unfavoured and ants search only in the neighbourhood of good solutions, thus limiting the exploration of promising heuristics in bad tours. By enforcing the same updating rule on all visited edges in an ant-based hyper-heuristic, no specific edges are dominant, thus giving fair choices for the ants to select promising edges. In the next chapter, to investigate the generality of our approach, we apply our approach to another routing problem, the TSP.

Chapter 6

Application to the Travelling Salesman Problem (TSP)

6.0 Introduction

In the previous chapter, we have applied the ant-based hyper-heuristic to the capacitated vehicle routing problem (CVRP). The algorithm is seen to be effective for solving a variety of CVRP instances. In this chapter, we aim to investigate the generality of our approach to the travelling salesman problem (TSP). To our knowledge, ant-based hyper-heuristic have never been applied to the TSP. TSP is the problem of finding the shortest or cheapest way of visiting all cities in a tour and returning to the starting city. Related work with respect to the TSP is presented in chapter 3. This chapter is structured as follows; section 6.1 describes the problem. Section 6.2 presents the experimental setup we use for the TSP. Section 6.3 describes the low level heuristics utilised and section 6.4 presents the computational experiments. Section 6.5 concludes the chapter.

6.1 Problem Formulation

The TSP is to find a tour for a given number of cities, to visit the cities once and returning to the starting city. The objective of the problem is to minimise the tour length. Some solution methods for TSP are discussed in chapter 3. The problem can be defined as follows:

A salesman is required to visit a sequence of cities 1, ..., n. He will start at an initial city, visits each city once and returns back to the initial city. Let d_{ij} ($i \neq j$, = 0, 1, 2, ..., n) be the tour length for the problem. We want to minimise the tour length. The mathematical formulation as adapted from Bryant and Benjamin (2000) is presented below:

$$\Sigma_{0 \leq i} \ \Sigma_{j \leq n} \ d_{ij} x_{ij}$$

Subject to

$$\Sigma_{i=o \ i \neq j}^{n} x_{ij} = 1 \qquad (j = 1, ..., n)$$

$$\Sigma_{j=o \ j \neq i}^{n} x_{ij} = 1 \qquad (i = 1, ..., n)$$

where x_{ij} are non-negative integers. The constraint x_{ij} is required to be 1 so that the relationship between these two cities exists which can be expressed as: the salesman's tour from city i to city j if and only if $x_{ij} = 1$.

6.2 Experimental Setup

We implement our experiments on PC Pentium R 3.4 GHz with 1 GB RAM running on Microsoft Windows 2000. The ant-based hyper-heuristic is tested on well known datasets from the TSP library (The Travelling Salesman Library 2008). We tested our approach on instances of sizes n = (30, 76, 51, 100), where n = the number of cities. We chose these cities to enable us to make comparisons with the scientific literature. The initial solution is generated using the same sweep algorithm as we used for the CVRP problems (see section 3.2.1.1.2). The parameters (initial placement of the ants, the influence rate for pheromone values (α), the influence rate for visibility values (β) and the evaporation rate (ρ) are set to the same values as those used for the CVRP problem (chapter 5). We use the same values in order to test the generality of the algorithm. In all experiments, the approach is run for 1000 iterations, or until the algorithm fails to find an improvement for 100 iterations. The best result found is returned as the final solution.

In this chapter, we utilise 10 low level heuristics (we refer to these heuristics as TSP heuristics). They are based on 2-opt. Furthermore, we also include the 20 heuristics for the CVRP (we refer to these heuristics as CVRP heuristics; as in section 5.2) to further investigate the behaviour of our approach. The objectives of the experiments conducted in this chapter are the following:

- To demonstrate that the ant-based hyper-heuristic can be applied across different problem domains and instances.
- To investigate if the algorithm can select appropriate low level heuristics at each decision point.
- To demonstrate that the algorithm can produce competitive results when compared to other methods.

6.3 Low level heuristics

In this section, we developed ten simple low level heuristics for the TSP and most of them are based on 2-opt moves as they are easy to implement (***Tsp_h1*** - ***Tsp_h10***). The TSP heuristics are described below. The other set is the low level heuristics used for the CVRP problem, as described in section 5.2. CVRP heuristics operate on a tour in a single route or tours from different routes.

1. ***Tsp_h1***: Two cities are selected and part of the tour is reversed between these two selected cities. This is done on every city, starting from the initial city. For example, as below: city 2 and 7 were selected. The customers in between these cities will be reversed. Figure 6.1(a) and Figure 6.1(b) illustrate the example.

   Initial tour : 1 **2** 3 4 5 6 **7** 8

   Resulting tour : 1 **2** 6 5 4 3 **7** 8

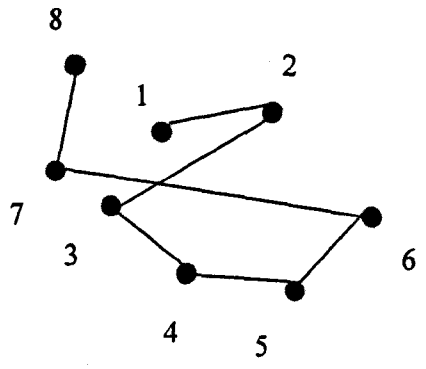

Figure 6.1(a): initial tour

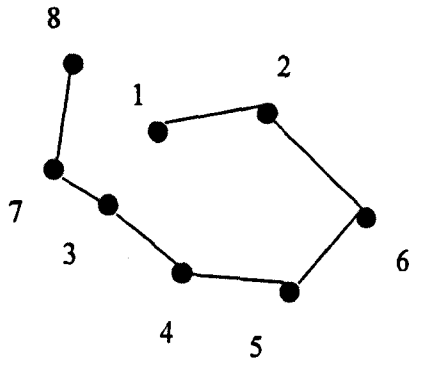

Figure 6.1(b): resulting tour

2. ***Tsp_h2***: 2 pair of cities will be selected from the tour. This is done on every city, starting from the initial city. For example, city and city 3 with city 2 and city 6 and city 7 are selected. 2-opt moves are done on the route. Figure 6.2(a) and Figure 6.2(b) illustrate the example.

Initial tour : 0 1 **2 3** 4 5 **6 7** 8 0

Resulting tour : 0 1 2 6 5 4 3 7 8 0

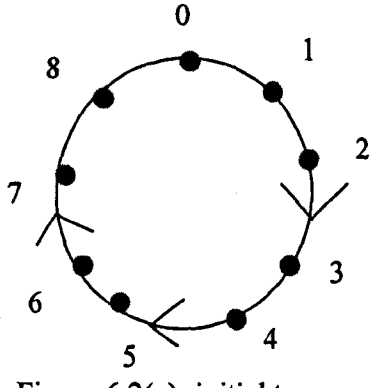

Figure 6.2(a): initial tour

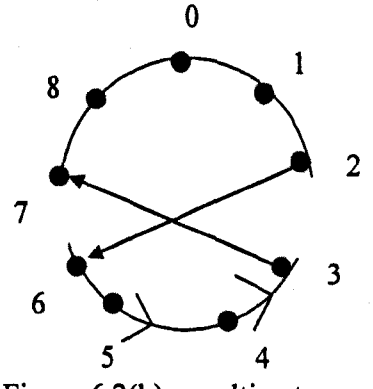

Figure 6.2(b): resulting tour

3. ***Tsp_h3***: 2-opt moves will be carried out, but with the cities being chosen randomly. The example is illustrated as above Figure 6.2(a) and Figure 6.2(b).

4. ***Tsp_h4***: Two cities are selected randomly and they will swap positions. For example; city 2 and 7 are selected. City 2 and 7 will swap positions. Figure 6.3(a) and Figure 6.3(b) illustrate the example.

Initial tour : 1 2 3 4 5 6 7 8

Resulting tour : 1 7 3 4 5 6 2 8

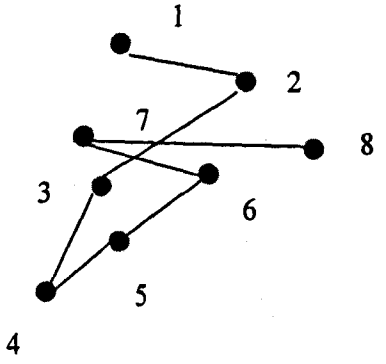

Figure 6.3(a): initial tour

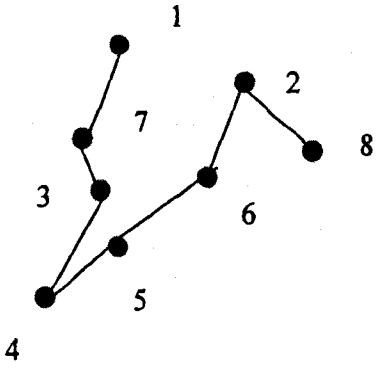

Figure 6.3(b): resulting tour

5. ***Tsp_h5***: Tsp_h5 operates the same as ***Tsp_h2***. The first move that produces an improvement over the current solution will be accepted.

6. ***Tsp_h6***: Two cities are selected and reverse a part of a tour between these two selected cities. The example is as in ***Tsp_h1***, however, the procedure is done on random selected cities.

7. ***Tsp_h7***: Two cities are selected randomly and they will swap positions. The example is as in ***Tsp_h4***. This procedure will repeatedly being applied until no further improvement is made.

8. ***Tsp_h8***: Two cities starting from two ends will be selected and swap positions. The in-between cities will be reversed. This procedure will be done on all cities. The example is as in ***Tsp_h2***. The best improvement will be recorded.

9. ***Tsp_h9***: Two cities starting from two ends will be selected and swap positions. This procedure will be done on all cities. The example is as in ***Tsp_h4***. The best improvement will be recorded.

10. ***Tsp_h10***: Two cities starting from two ends will be selected and the in-between cities will be reversed. This procedure will be done on all cities. The example is as in ***Tsp_h1***. The best improvement will be recorded.

6.4 Experiments

The aim of this section is to test the effectiveness of our approach. We utilize two different sets of low level heuristics in these experiments. The first set is the TSP heuristics which operates only on the tour produced. We apply our approach to the TSP instances from the TSP library (The Travelling Salesman Library 2008). In this experiment, we utilise 10 low levels heuristic (***Tsp_h1*** - ***Tsp_h10***); as described in section 6.3 and use 10 ants (equal to the number of low level heuristics) and initially placed the ants randomly on heuristics available on the network.

The second experiment utilises the second set of low level heuristics which are used for the CVRP problem (as described in section 5.2). CVRP heuristics operate on a tour in a single route or tours from different routes. The aim of this experiment is to investigate the behaviour of our approach when applied to CVRP low level heuristics. These experiments were run for 30 times with 1000 iterations each or until no improvement for 100 iterations. The best, worst, average results, time and standard deviation are recorded. The results are presented in Table 6.1.

Datasets	Best (TSP heuristics)	Worst (TSP heuristics)	Average (TSP heuristics)	Std Dev (TSP heuristics)	Average time (TSP heuristics)	Best (CVRP heuristics)	Worst (CVRP heuristics)	Average (CVRP heuristics)	Std Dev (CVRP heuristics)	Average time (CVRP heuristics)
Oliver30	**427.90**	429.43	429.07	0.58	10.92	**427.90**	429.90	429.27	0.44	8.94
Eil51	435.47	451.58	440.24	4.14	15.35	**434.85**	462.39	447.31	5.61	12.50
Eil76	**559.33**	585.09	569.56	7.19	47.81	569.12	595.00	584.94	5.79	34.49
KroA100	**21458.50**	23183.00	22039.33	393.93	95.59	21704.70	22593.30	22070.94	227.12	74.29

Table 6.1: Ant-based hyper-heuristic - Experiments with different low level heuristics (TSP low level heuristics and CVRP low level heuristics)

From the results, it can be seen that TSP heuristics perform better than CVRP heuristics in two datasets. For dataset 1 (Oliver30) both heuristics performed equally, however, in terms of average time and standard deviation, the CVRP heuristics have performed better. The best results for dataset 3 (Eil76) and dataset 4 (KroA100) are obtained by TSP heuristics which are based on 2-opt procedure, operating on a single tour. The average results obtained by TSP heuristics also demonstrate superior results than those obtained by CVRP heuristics. However, when compared to computational time, CVRP heuristics appear to take less time than the TSP heuristics. This is probably due to less heuristics are appropriate to explore the search space for the TSP (many of CVRP heuristics operate on different route/tour; TSP problem, however, only involves one route/tour). Furthermore, we also conduct statistical tests (t-test) to further analyse the similarity of these experiments. The null hypotheses H_0 , (E_1 is not significantly different to E_2) are tested. E_1 are experiments for TSP heuristics and E_2 are experiments for the CVRP heuristics. We perform a statistical test (t-test) with 95% confidence level for each datasets and the results are presented in Table 6.2. The data used for these tests are shown in Appendix H and Appendix I.

E_1 / E_2	**Oliver30**	**Eil51**	**Eil76**	**KroA100**
Oliver30	accept H_0	X	X	X
Eil51	X	reject H_0	X	X
Eil76	X	X	reject H_0	X
KroA100	X	X	X	accept H_0

Table 6.2: T-test for different categories of low level heuristics, E_1 (TSP heuristics) and E_2 (CVRP heuristics)

The findings in Table 6.2 show that for Oliver30 dataset and KroA100, we accept H_0 and for the Eil51 and Eil76 datasets, the results are found to be significantly different where we reject the null hypotheses H_0 for both heuristics categories. From the table, we observe that for small instance (30 cities) and large instance (100 cities), both TSP and CVRP heuristics do not perform any differences. For medium instances (51 cities and 76 cities), the approach has shown differences for both datasets. Therefore, it can be seen that for medium instances, our approach shows better consistency in identifying appropriate heuristics. By using any datasets, ant-based hyper-heuristic is seen to be able to produce competitive results. From this observation, the ant-based hyper-heuristic has demonstrated that it is a reusable method; given different sets of low level heuristics for the same class of problem.

We further conducted the third experiment which aims to make comparisons between ant-based hyper-heuristic and ACO hyper-heuristic on TSP instances. ACO hyper-heuristic are different from the ant-based hyper-heuristic in terms of pheromone updating procedure. Detail descriptions are presented in section 4.7. In this experiment, we use the TSP heuristics. The results are presented in Table 6.3.

Datasets	Best Anthh	Worst Anthh	Average Anthh	Std Dev Anthh	Average time Anthh	Best ACOhh	Worst ACOhh	Average ACOhh	Std Dev ACOhh	Average time ACOhh
Oliver30	**427.90**	429.43	429.07	0.58	10.92	**427.90**	429.90	429.46	0.48	9.37
Eil51	435.47	451.58	440.24	4.14	15.35	**429.70**	463.45	439.41	9.84	52.19
Eil76	**559.33**	585.09	569.56	7.19	47.81	568.89	594.26	582.41	6.33	160.92
KroA100	**21458.50**	23183.00	22039.33	393.93	95.59	21704.70	22562.50	22069.16	223.01	74.48

Table 6.3: Comparisons between ACO hyper-heuristic and ant-based hyper-heuristic: Experiments with TSP low level heuristics

Observations of the table show that the best results for two out of four datasets are produced by the ant-based hyper-heuristic. For dataset 1 (Oliver30) both approaches perform the same. This indicates that modifications on the process of updating the pheromone value do not have an influence to guide the hyper-heuristic to pick better heuristics. This is in line with our experiments in the previous chapter (section 5.10). Further null hypotheses H_0 are tested to ascertain these effects and listed below:

Null hypotheses, H_0 represent E_1 is not significantly different from E_2. E_1 is the experiments for ACO hyper-heuristic and E_2 is the experiments for the ant-based hyper-heuristic. We perform a statistical test (t-test) with 95% confidence level for each dataset and the results are presented in Table 6.4. The data used for these tests are shown in Appendix G and Appendix I.

E_1 / E_2	**Oliver30**	**Eil51**	**Eil76**	**KroA100**
Oliver30	reject H_0	X	X	X
Eil51	X	accept H_0	X	X
Eil76	X	X	reject H_0	X
KroA100	X	X	X	accept H_0

Table 6.4: T-test for E_1 (ACO hyper-heuristic) and E_2 (ant-based hyper-heuristic)

From the above table, for datasets Oliver30 and Eil76, the null hypotheses H_0 are rejected which shows that there is a difference between these experiments, which indicates that modifications on

local and global updates for the pheromone values for ACO hyper-heuristic do have some influence on the solutions produced for these datasets. However, the results of ACO hyper-heuristic are not better than the ant-based hyper-heuristic. Datasets Eil51 and KroA100 are found to have no significant difference in these two experiments, thus indicating that the modifications do not have an influence in guiding the ACO hyper-heuristic to search for better solutions. It appears that in dataset Eil51, the result produced is better (1.3%) than the ant-based hyper-heuristic. However, the worst solutions for ant-based hyper-heuristic are better compared to ACO hyper-heuristic. The time and standard deviation required to obtain a solution is also better for ant-based hyper-heuristic. This shows that ant-based hyper-heuristic demonstrate better consistency than the ACO hyper-heuristic. As shown in the experiments, modifying the process of updating the pheromone values does not result in giving good solutions for the TSP although for dataset Eil51, ACO hyper-heuristic produce better solutions.

In our next work, we compare our results to several other methods. The methods are ant colonies (ACS), genetic algorithm (GA), evolutionary programming (EP), tabu search (TS), adaptive tabu search (ATS) simulated annealing (SA) and annealing-genetic algorithm (AG). The results are presented in Table 6.5. The best known results for these datasets are obtained from The Traveling Salesman Library (2008). Results for ACS, EP, SA and AG are obtained from Dorigo and Gambardella (1997). Results for GA, TS and ATS are obtained from Suwannarongsri and Puangdownreong (2012). The results for best known, SA and AG are presented in integer values.

Datasets	Anthh	ACS	GA	EP	TS	ATS	SA	AG	Best known
Oliver30	427.90	423.74	N/A	423.74	N/A	N/A	424	420	424
Eil51	435.47	427.96	441.46	427.86	445.05	438.12	443	436	426
Eil76	559.33	542.31	548.26	549.18	582.62	540.17	580	561	538
KroA100	21458.50	21285.44	22875.42	N/A	23664.18	21526.56	N/A	N/A	21282

Table 6.5: Comparison of ant-based hyper-heuristic to other methods.

Our results are found to be competitive to other methods although they do not produce any best known solution. We observed for Eil51 and Eil76 dataset, the result is better than simulated annealing (SA) and annealing-genetic (AG) algorithm. For KroA100 dataset, the result is better than genetic algorithm (GA), tabu search (TS) and adaptive tabu search (ATS). Therefore, we conclude that ant-based hyper-heuristic are comparable to other problem specific methods.

6.5 Summary

In order to test the generality of an ant-based hyper-heuristic, we applied it to another routing problem; the TSP. TSP is a *NP*-hard problem and has been one of the most widely studied areas for research in combinatorial problems. Numerous methods have been applied to TSP such as heuristics and meta-heuristics. However, to our knowledge, hyper-heuristic has not been applied to the problem. We investigate the performance of ant-based hyper-heuristic and it appears that it can solve four different instances and capable of producing competitive results with results published in the literature. To further investigate the behaviour of the algorithm, we apply it to two different sets of low level heuristics; the TSP heuristics and the CVRP heuristics. It appears the TSP heuristics have outperformed the CVRP heuristics for two out of four datasets. However the differences are very small. The time consumed by the CVRP heuristics is less than the TSP heuristics and it is also has very minimal differences. From the observation, we suggest that the ant-based hyper-heuristic is a reusable method due to its capability to select appropriate low level heuristics; given heuristics for the same class of problems.

The ACO hyper-heuristic were developed to observe the effect of local and global updating of pheromone values. The comparison was made and it appears that this activity does not give any influence on the performance of the algorithm. Although the ant-based hyper-heuristic does not produce any best known results in the literature, it is able to outperform some other popular methods. The advantage of this approach is it does not need any parameter tuning and the low level heuristics utilised are simple and easy to implement.

Chapter 7

Conclusions and Future Work

7.0 Introduction

This thesis describes the development of a more general approach based on ant algorithms. The idea of this approach is to provide a way to combine few simple heuristics that can solve different optimisation problems across different instances without extensive parameter tuning and by using simple and easy to implement low level heuristics. In an ant algorithm, the ability to find solution is guided by pheromone and visibility values (heuristic information). This process thus involves determining ants as hyper-heuristic agents to make decision of which heuristics to choose at each decision point and how to present the pheromone and visibility values for the approach. Previous ant algorithm hyper-heuristic in Burke et al. (2003c) and Chen et al. (2007) are analysed and the key properties and functions are identified. As a result, a new ant-based hyper-heuristic is developed. The resultant approach will be applied to two routing problems of which, to our best knowledge, have not been applied to any routing problems.

7.1 Evaluation of the Aims

This thesis is concerned with the investigation of ant-based hyper-heuristic. We apply the approach to two routing problems; the capacitated vehicle routing problem (CVRP) and the travelling salesman problem (TSP). In section 1.1 (chapter 1), the aims and objectives of the thesis are outlined into four parts. In this section, these objectives will be evaluated based on the work presented in this thesis.

Placing our work in the context of previous work

The first objective is to investigate the concepts and background for the work done in this thesis. Chapter 2 is divided into four sections. In the first section (section 2.1), the definition of combinatorial optimisation problem, the algorithm complexity; how the efficiency of an algorithm is measured and the *NP* complete classes are presented. In order to understand the solution methods for optimisation problems, a few solution methods are presented in the second section. Problem specific solution methods are reviewed to identify how these approaches construct and improve solutions, thus finding ways to escape from local optimum. Among the methods discussed are genetic algorithms, tabu search, simulated annealing, greedy randomised adaptive search (GRASP) and neighbourhood search (VNS). In the third section (section 2.3.6), ant algorithms are highlighted in some detail; which is based on the observation of how ants are able to find the shortest path between food sources and their nest, further adapting this bahaviour to construct solutions. The different versions of ant algorithms; the ant system, elitist ant, rank-based ant, max-min ant and ant colony system are presented. As a result, it is determined that

pheromone and visibility (heuristic information) are two important factors in the process of decision making for the algorithm. In the fourth section (section 2.4), the concept of hyper-heuristic and their previous work are then explored. The difference between two categories of hyper-heuristic; the heuristic selection and the heuristic generation are presented. Furthermore, two categories of hyper-heuristic methods; constructive hyper-heuristic and perturbative hyper-heuristic are discussed. In contrast to metaheuristics approaches, hyper-heuristic should consider to solve different problems and instances. Several hyper-heuristic approaches; focusing on the heuristic selection methodology, are presented. In section 2.5, ant algorithm hyper-heuristic are outlined in detail; the properties and functions of two ant hyper-heuristics (Burke et al. 2005b, Chen et al. 2007) are investigated. Such properties include the mechanism of choosing heuristics, pheromone and visibility update are explained. As opposed to an ant algorithm, this method should consider the non-domain specific knowledge to represent the pheromone and visibility values. The fifth section (chapter 3), the routing problems, the VRP and TSP which will be the problem domain tested for the proposed algorithm are examined. Various variants of VRP were presented and they differ from each other, either constrained by the limit of capacity or time windows. Several solution methods on CVRP are outlined. The TSP is further reviewed; heuristics to construct solution and heuristics to improve the solution are discussed.

Establishing the hyper-heuristic algorithm based on ant algorithms

Heuristics and metaheuristics often use problem specific knowledge to build or improve solutions. They have been shown to work well on certain instances. For other instances, however, they do not perform well and often, they are expensive to adapt to new instances and

problems. Furthermore, they are often time-consuming and involve knowledge intensive process that requires deep understanding of the problem domain. This motivates the investigation of developing a methodology (hyper-heuristic) that can produce good quality solutions across different instances and problems and which do not require extensive parameter tuning. The main goal of hyper-heuristic is to produce more generally applicable search methodologies. In this thesis, we develop a generalised heuristic method (hyper-heuristic) based on ant algorithms (Dorigo et al. 1991, Dorigo et al. 1996) and ant algorithm hyper-heuristic (Burke et al. 2003c, Chen et al. 2007). What inspire us is the interesting behaviour of the ants; in their ability to find the shortest path between their nests and the food source. Hyper-heuristic can be classified in two ways; the heuristic generation and the heuristic selection. Our work is concerned with heuristic selection methods, where heuristics are used to choose heuristics

In ant algorithms, a given problem is represented as directed graph and the set of nodes is the candidate solutions to a given problem. Each ant will perform a tour by visiting a sequence of nodes by selecting a node after another guided by some information; the pheromone and the visibility values. The pheromone and visibility values rely on the problem specific knowledge of the problem being solved. In our approach, in order to produce a generalized approach, the pheromone and visibility values should consider a non-domain specific knowledge where the amount of pheromone corresponds to the quality of the solutions found by the ants, where we take the improvement made between two heuristics i and j as the quality of the solution. In existing ant algorithm hyper-heuristic (Burke et al. 2003c, Chen et al. 2007), the amount of pheromone corresponds to the quality of the solution found by the ants. To non-performing heuristics, however, no pheromone is rewarded. In this thesis, we propose to provide all visited

heuristics with some amount of pheromone. The distribution of pheromone values will be distributed proportioned to the performance done by the ants. This is to encourage the exploration of new edges that might lead to better solutions. Visibility values are measured as to how well two heuristics work together. This is represented as the computational time between two heuristics *i* and *j*. A heuristic is allowed to be visited as many times as possible compared to the original ant algorithms where a heuristic is prohibited to be visited more than once. Further ACO hyper-heuristic is developed. The difference between the first approach and ACO hyper-heuristic is in the context of pheromone updating procedure. ACO uses the global and local updating rule. The global update uses edges that produce the best solution found at a current iteration to update the pheromone trail. This is to encourage the use of good-performing path and to increase the probability of choosing the specific path. The local update is performed after each ant performs a tour. This is to ensure that the pheromone values do not go unbounded. The aim is to determine competitive solutions as well as to investigate the generalisation of the proposed algorithms when applied to these two different routing problems.

Implementing our approach on the capacitated vehicle routing problem (CVRP)

Our approach for the first time is applied to capacitated vehicle routing problem (CVRP). To our best knowledge, at the time this thesis is written, the approach has not been applied to any capacitated vehicle routing problems (CVRP). Ant algorithms have been applied to CVRP (Bullnheimer et al. 1999a, 1999c) and have produced competitive results. Therefore, it is assumed that there is a reasonable possibility that ant-based hyper-heuristic could perform well for the problem.

Initially, the setting of parameters was established to determine the appropriate values. The work proposes three different parameter settings; the starting position, the pheromone and visibility rates and finally the evaporation rate values. Furthermore, to demonstrate the effectiveness of the ant-based hyper-heuristic, series of experiments are conducted. Below are some observations and conclusions from the work. The ant-based hyper-heuristic uses 20 different low level heuristics, which utilises 2-opt moves with simple move and swap neighbourhood moves. We test our approach with different number of low level heuristics. The experiment demonstrates that higher number of heuristics used will guide the ants to generate better solutions. Statistical tests also show that ant-based hyper-heuristic is statistically different from random hyper-heuristic which demonstrate that there is possibility that ants are guided by pheromone and visibility; in contrast with random hyper-heuristic which selects heuristics at random. We believe that this shows ant-based hyper-heuristic incorporates some learning mechanisms.

The experimental results on CVRP have shown that it is competitive with other methods although it does not produce any best known results. Most importantly, it demonstrates how simple and easy it is to implement low level heuristics, with no extensive parameter tuning. Our approach performs better than the ant system for datasets with 75, 100(a), 150 and 199 customers. For datasets with 120 and 199 customers, we produce better results than the simulated annealing algorithm. For random distributed problems (problems: 1,2,3,6,7), we are able to produce solutions that have average deviation less than 6% from best known solutions while for clustered problems (problem: 4,5), our approach only manage to produce solutions with average deviation less than 11%.

The comparison between the ACO hyper-heuristic further shows that ant-based hyper-heuristic perform better despite the sophisticated procedure of pheromone updating in ACO hyper-heuristic. This is inconsistent with the performance of ACO algorithm in the literature where by modifying the process of updating the pheromone values, the results are better (Dorigo and Di Caro 1999c). Heuristics perform differently at each decision point. Bad tours become highly unfavoured and ants search only in the neighbourhood of good solutions thus limiting the exploration of promising heuristics in bad tours. Therefore, it can be seen that enforcing global and local updates of pheromone values does not appear to contribute to any success in obtaining better results than ant-based hyper-heuristic

Investigating the generality of our approach by applying it on another routing problem, the Travelling Salesman Problem (TSP).

To test the generality of our approach, we apply it to another routing problem; the TSP. Our results are found to be competitive compared to other methods although they do not produce any best known solutions. We observe that for Eil51 and Eil76 dataset, the results are better than simulated annealing (SA) and annealing-genetic (AG) algorithm. For KroA100 dataset, the results are better than genetic algorithm (GA). In addition, we test the effectiveness of our approach by utilising two different sets of low level heuristics (TSP heuristics and CVRP heuristics). The average results obtained by TSP heuristics also demonstrate superior results than those obtained by CVRP heuristics. However, when compared to computational time, CVRP heuristics appear to take less time than the TSP heuristics. This is probably due to less heuristics are appropriate to explore the search space for the TSP (many CVRP heuristics operate on

different routes/tours; TSP problem, however, only involves one route/tour). From this observation, it can be seen that by using any low level heuristics appropriate for the same class of problems, ant-based hyper-heuristic is seen to be able to produce competitive results. This has demonstrated that the ant-based hyper-heuristic is a reusable method.

ACO hyper-heuristic is further applied to TSP problems which aim to make comparisons between ant-based hyper-heuristic and ACO hyper-heuristic on TSP instances. Observations of these results show that best solutions for two out of four datasets are produced by ant-based hyper-heuristic. For dataset 1 (Oliver30) both approaches perform the same. This is in line with our experiments in the previous application in CVRP which indicates that modifications on the process of updating the pheromone values do not have any influence to guide the hyper-heuristic to pick better heuristics.

7.2 Contributions

The work carried out in this thesis has the following contributions:

- We had developed a hyper-heuristic methodology based on ant algorithm. We introduced a new pheromone and visibility update rule; which proved to produce a better solution than the previous two ant algorithm hyper-heuristic (Burke et al. 2003c and Chen et al. 2007). In previous ant algorithms hyper-heuristics, after ants have visited a certain number of heuristics, they paused to analyse the edges and lay some amount of pheromone according to improvements of the solutions. Only edges that contributed to

improvement to current solution are rewarded with pheromone whereas in our approach, all visited edges will be given some amount of pheromone. The distribution of pheromone values will be distributed proportioned to the performance done by the ants. This is to ensure that all visited edges will be given pheromone values according to their performance. Visibility values are represented as computational time between two heuristics *i* and *j*. This has shown that our approach has further improved the solutions of previous ant algorithms hyper-heuristic.

- We had developed ACO hyper-heuristic which had new pheromone updating rule as in ACO algorithm. We had tested it to the problems and it showed that it did not contribute to any better solutions. This was inconsistent with the performance of ACO algorithm in the literature (Dorigo and Di Caro 1999c).

- Our approach had shown to have reusable methods; and using simple and easy to implement low level heuristics can produce competitive solutions compared to other problem specific approaches. To our best knowledge, at the time we wrote this thesis, it presented first investigation of ant-based hyper-heuristic on two routing problems; the Capacitated Vehicle Routing and the Travelling Salesman Problem.

7.3 Strengths and Limitation of the Algorithm

The strengths and limitations of our approach are listed below:

- The approach is able to produce competitive results in both CVRP and TSP problems by using simple moves and swap procedure and without extensive parameter tuning. The ant-based hyper-heuristic shows the capabilities of having a learning mechanism which is guided by the visibility and the pheromone values. We have compared it with the random hyper-heuristic which has no learning mechanism.
- Although using different sets of low level heuristics for the same class of problem (CVRP and TSP), the approach has demonstrated its reusability.
- The limitation of this approach is the amount of time needed to produce solutions for larger instances. This is probably due to the number of ants utilised to search for solutions. Decreasing the number of ants might lead to promising search space not being explored.

7.4 Future Work

The following suggestions for future work are given:

- In this thesis, we develop the ant-based hyper-heuristic that adapt the ant algorithm (ant system) behaviour. The approach produces good results however, it does not produce any best known results for both problems (CVRP and TSP). It is worthwhile that other variant of ant algorithms being investigated in order to produce better competitive results. Among the variants that are worth investigating is elitist ant system, ant system rank-based and max-min ant system. Instead of enforcing the global and local pheromone updates as in ACO hyper-heuristic which do not contribute to any improvement, it may be worth to investigate the effects of sorting the ants according to their performance and pheromone is deposited according to the rank of the ants. Further investigation would be to introduce the upper and lower boundaries of the pheromone trails. This might help to obtain better quality solutions.

- The drawback of this approach is the amount of time needed to produce solutions for larger instances. It would be beneficial to extend the work by investigating ways to improve the computational time needed to produce solutions; especially on larger instances. It is worthwhile that the number of ants utilised being investigated. Possibly the idea of introducing multiple colonies of ants that have separate pheromone deposits for each colony could also be investigated. This would consider the benefit of using

separate pheromone trails which result in the need to separate the most likely edge to better solutions.

- The parameter setting for this approach is set to be static throughout all experiments. It is worth investigating if these parameters continue to be adaptive (dynamic) based on the information obtained while the approach is being executed. This will improve the performance of the approach.

- Hyper-heuristic produces solutions that are soon enough, good enough, cheap enough for variety of problems and problem instances. In most industries, problem owners would prefer simple to-implement problem solving methods that could solve problems quickly enough. In this thesis, we utilise simple move and swap low level heuristics. However, it would be interesting to compare the performance of these heuristics to several complex intelligent low level heuristics.

- Finally, we would like to apply this ant-based hyper-heuristic to other problems to demonstrate the generality of the approach and classify which problem domains that ant-based hyper-heuristic seems to work well and for which they do not.

To conclude this chapter, it is interesting to note that with a single method, several optimisation problems can be solved, without having to deal with extensive parameter tuning. We hope that in the future, more generalised method are investigated to raise the level of generality of a search method.

Appendix A

Results for different starting positions (heuristic1, each heuristic, random)

Heuristic 1	Each heuristic	Random
868.49	896.78	877.4
859.03	854.95	857.9
864.05	909.82	877.4
882.21	868.81	867.9
885.38	860.21	864.0
877.37	865.52	857.9
867.97	878.97	857.9
878.97	885.38	879.0
867.87	896.78	842.3
872.08	865.52	864.0
857.91	857.91	867.9
877.37	877.37	857.9
896.78	878.97	867.9
867.87	885.38	857.9
909.82	878.97	868.8
882.21	885.38	868.8
885.38	916.00	859.0
877.37	877.37	879.6
867.87	896.78	857.9
878.97	878.97	857.9
885.38	878.97	877.4
909.82	877.37	857.9
867.87	916.00	868.8
896.78	882.21	885.4
965.80	878.97	877.4
909.82	877.37	879.0
882.21	896.78	867.9
868.81	882.21	879.0
885.38	896.78	868.8
909.82	882.21	857.9

Appendix B

Results for different pheromone and visibility rates

pheromone = 0.1 visibility = 0.9	pheromone = 0.2 visibility = 0.8	pheromone = 0.3 visibility = 0.7
solution	solution	solution
860.33	864.101	872.751
860.29	861.582	875.226
870.667	869.782	849.186
888.139	873.709	864.557
866.464	869.346	878.738
865.262	869.893	878.019
878.974	864.542	892.021
864.064	868.028	870.478
883.75	862.77	868.674
868.358	865.948	870.049
878.974	869.346	883.75
869.782	862.77	870.667
878.974	869.782	873.709
865.262	869.346	865.262
883.75	864.064	873.709
865.262	869.346	864.064
861.582	873.709	868.358
868.358	865.262	869.346
883.75	869.782	864.064
864.064	864.064	865.262
878.738	860.33	869.782
867.793	860.29	865.262
865.262	862.77	881.398
881.398	873.709	869.782
869.782	869.346	881.398
881.398	865.262	873.709
869.539	864.064	869.346
865.326	862.77	878.974
868.358	869.782	869.893
864.064	873.709	873.709

pheromone = 0.4 visibility = 0.6	pheromone = 0.5 visibility = 0.5	pheromone = 0.6 visibility = 0.4
solution	solution	solution
883.606	858.483	888.299
908.714	864.177	880.076
875.185	859.908	873.399
887.763	870.812	864.95
883.47	875.547	881.398
861.097	865.487	869.539
858.853	859.925	865.326
881.066	860.421	868.661
859.69	865.653	878.738
880.92	852.434	867.793
881.066	852.788	865.326
861.097	875.852	873.709
867.793	862.608	881.398
881.066	877.978	869.539
878.738	844.67	869.782
880.92	853.718	867.793
880.92	866.724	869.346
864.064	860.979	888.299
875.185	842.34	865.262
887.763	863.448	864.95
865.262	874.206	881.398
881.066	859.237	869.782
878.738	874.501	864.064
865.262	878.426	864.064
881.398	865.96	881.398
869.782	851.286	867.793
867.793	879.60	869.539
880.92	863.215	865.326
883.47	872.659	880.076
869.782	854.973	867.793

pheromone = 0.7 visibility = 0.3	pheromone = 0.8 visibility = 0.2	pheromone = 0.9 visibility = 0.1
solution	solution	solution
871.846	879.91	897.526
872.099	864.14	866.594
878.881	869.724	859.708
859.125	881.649	868.519
871.863	876.892	871.318
870.071	865.666	859.552
872.405	879.147	872.186
867.812	855.127	880.644
868.595	885.608	875.384
865.176	866.572	877.171
869.539	865.666	876.892
869.782	876.892	879.147
868.595	879.147	877.171
873.399	885.608	885.608
867.793	865.262	868.519
872.405	881.066	879.147
864.95	876.892	876.892
867.793	859.69	865.666
865.326	880.92	881.066
868.595	855.127	859.708
873.399	876.892	879.147
867.812	866.572	871.318
868.595	879.147	876.892
865.176	855.127	872.186
867.793	879.147	879.147
865.326	865.262	859.552
873.709	881.066	885.608
869.782	865.666	876.892
867.793	865.262	872.186
865.326	879.147	868.519

Appendix C

Results for different evaporation rates

0.1	0.2	0.3	0.4	0.5
876.49	867.60	876.86	864.09	865.84
860.37	874.28	870.18	873.49	855.37
870.71	850.27	871.52	860.48	880.47
846.49	861.66	866.69	863.38	874.28
853.89	865.42	872.86	866.60	847.14
863.44	867.03	865.84	859.75	869.37
875.39	874.27	843.63	864.21	853.89
848.60	873.87	868.29	881.51	879.55
849.03	855.37	868.31	862.01	860.48
880.74	884.79	876.55	880.43	872.41

0.6	0.7	0.8	0.9
863.65	872.85	878.64	880.57
867.24	890.12	879.62	868.67
880.47	869.37	868.91	851.31
864.81	873.44	872.58	864.88
913.80	891.42	873.84	879.25
872.41	882.89	864.70	872.66
901.05	866.85	886.57	869.58
867.31	872.22	875.47	879.20
878.73	870.64	875.25	876.18
862.82	881.38	862.54	857.24

Appendix D

Results for different combination number of low level heuristics

10 good	time	10 bad	time	10 intuitive	time	20 combine	time
884.126	10.28	894.641	3.23	866.544	27.82	858.483	41.40
875.576	31.59	966.013	2.50	868.411	18.43	864.177	51.96
885.981	12.26	901.516	1.86	879.026	11.01	859.908	20.19
867.271	23.46	918.523	3.29	876.108	6.09	870.812	22.98
916.237	10.47	924.595	3.71	871.081	9.87	875.547	29.62
889.12	14.10	879.827	4.97	860.253	12.84	865.487	16.89
865.489	13.98	896.58	4.96	868.604	17.26	859.925	31.11
880.469	33.69	922.881	4.65	878.159	5.33	860.421	34.46
884.978	26.36	902.99	2.96	889.291	11.32	865.653	24.67
889.053	14.51	923.601	2.67	847.835	13.19	852.434	26.91
887.408	10.42	920.383	1.80	860.253	13.04	852.788	21.14
888.818	12.98	931.791	6.05	875.496	8.79	875.852	16.54
891.774	25.52	921.994	3.78	873.885	16.24	862.608	22.64
895.088	43.75	951.741	1.59	880.195	13.67	877.978	41.68
889.517	12.97	931.879	1.92	873.907	14.52	844.67	19.51
872.722	12.21	886.969	4.66	862.707	12.69	853.718	29.80
885.237	10.32	933.768	1.17	866.662	7.72	866.724	13.51
874.583	15.15	934.71	2.62	838.653	17.98	860.979	25.35
872.879	30.82	907.705	1.95	859.114	18.02	842.34	17.74
906.662	9.19	916.225	3.70	861.95	13.08	863.448	19.67
899.209	14.31	884.152	3.30	881.363	9.28	874.206	11.48
875.492	8.55	986.109	2.33	877.475	11.91	859.237	25.04
873.835	14.58	933.605	2.63	877.763	8.44	874.501	13.34
884.529	26.94	908.813	2.29	879.442	15.95	878.426	18.49
881.343	11.14	922.608	1.34	878.658	6.13	865.96	37.21
878.101	43.80	881.591	3.33	887.62	11.08	851.286	18.61
888.277	14.60	908.14	2.92	879.702	11.25	879.6	32.45
866.062	23.51	961.125	1.70	863.171	12.90	863.215	28.03
907.128	11.49	950.964	2.56	867.34	12.38	872.659	24.61
863.963	24.18	903.934	1.85	882.425	6.74	854.973	38.40

Appendix E

Results for Random Hyper-heuristic on CVRP

50 customers	time	75 customers	time	100(a) customers	time	100(b) customers	time
561.904	4.10	880.89	10.94	869.28	38.16	911.649	32.26
566.869	6.05	936.93	4.81	884.88	13.41	968.135	9.93
544.332	5.37	889.65	6.51	894.20	11.33	986.122	9.47
550.377	5.96	883.22	12.61	896.82	9.09	937.474	25.94
549.922	4.41	894.81	5.57	888.76	17.71	959.669	11.13
543.594	2.84	937.16	9.38	866.70	31.83	955.555	13.38
566.396	2.71	897.26	9.79	885.64	14.12	987.051	10.89
542.819	2.77	870.34	10.56	870.38	13.59	967.27	12.28
584.59	3.16	881.71	5.34	889.09	20.62	948.34	17.97
564.54	4.15	894.90	7.82	885.05	28.48	986.784	4.59
556.722	4.77	928.88	11.43	876.51	27.42	919.324	10.99
542.819	4.54	944.62	4.70	876.99	26.06	965.001	11.01
562.404	2.29	910.54	5.23	909.25	30.32	987.314	8.59
581.133	4.77	899.40	3.61	878.67	35.43	975.339	15.95
543.594	2.64	935.06	6.09	909.83	14.96	948.758	8.81
543.594	3.23	928.15	3.33	887.55	19.31	918.348	19.16
562.74	4.97	906.28	5.66	869.34	49.79	967.227	21.63
547.585	2.69	893.68	11.80	864.81	49.26	986.083	6.88
543.594	3.00	875.49	7.66	868.54	27.70	960.882	6.02
556.106	2.25	899.08	8.56	920.51	26.71	971.366	5.11
566.962	3.70	921.59	11.51	885.98	28.83	943.495	6.77
547.488	5.66	903.69	7.22	900.21	12.63	981.08	16.65
547.896	4.27	944.39	4.07	904.04	10.60	974.514	7.53
554.564	2.81	898.44	4.71	899.72	25.13	971.323	4.61
574.502	2.80	890.71	6.38	912.30	17.33	984.486	6.72
555.731	4.43	914.84	6.66	864.98	22.54	958.682	6.21
542.819	3.57	949.80	6.42	886.84	23.52	946.051	17.65
539.681	3.53	886.13	11.71	923.81	16.41	958.736	10.99
542.819	3.01	942.80	8.58	867.15	35.91	981.449	17.87
543.594	6.09	896.59	5.04	870.35	20.56	945.060	8.19

120 customers	time	150 customers	time	199 customers	time
1242.57	29.71	1105.17	70.24	1409.00	56.09
1154.22	48.33	1112.02	24.78	1387.98	138.66
1230.53	24.90	1101.28	35.12	1397.26	52.95
1179.52	67.80	1097.10	14.68	1408.88	73.24
1225.67	45.96	1105.23	35.50	1415.98	56.55
1165.37	33.16	1132.69	51.02	1391.21	74.81
1208.93	60.86	1094.69	14.17	1416.95	123.38
1207.75	28.70	1114.63	37.71	1442.78	53.17
1213.34	62.30	1112.11	100.35	1421.05	94.00
1184.91	47.66	1106.57	46.90	1413.55	1296.33
1235.97	29.66	1108.42	33.30	1398.73	51.88
1184.88	54.84	1084.77	45.75	1419.40	30.57
1145.61	66.90	1090.83	23.35	1413.90	26.11
1214.55	39.75	1092.85	51.87	1407.09	27.77
1198.57	71.08	1086.52	39.79	1430.43	25.34
1140.54	29.43	1155.46	21.44	1446.30	54.66
1244.22	33.79	1103.46	39.72	1416.50	58.10
1230.29	18.83	1114.77	19.52	1413.50	109.91
1165.43	23.89	1088.35	36.63	1403.60	70.39
1221.15	24.09	1100.83	39.77	1442.38	54.00
1206.45	23.77	1114.26	27.78	1449.68	34.50
1239.60	23.48	1103.65	43.08	1438.98	41.28
1221.11	27.55	1088.18	48.42	1430.24	163.01
1160.69	28.47	1107.32	17.08	1411.54	36.86
1190.51	30.60	1103.93	61.09	1403.87	62.64
1260.60	32.61	1109.35	49.05	1404.65	36.80
1152.69	56.01	1088.32	66.46	1436.59	77.52
1218.31	28.23	1092.47	31.58	1491.16	19.37
1208.320	26.52	1109.43	26.77	1405.31	114.59
1223.000	30.58	1087.84	28.25	1430.59	58.93

Appendix F

Results for Ant-based Hyper-heuristic for CVRP

50 customers	time	75 customers	time	100(a) customers	time	100(b) customers	time
539.681	3.22	867.35	6.04	858.483	41.40	927.749	15.72
532.996	10.73	880.64	7.69	864.177	51.96	976.538	15.76
542.639	4.84	871.78	6.10	859.908	20.19	949.324	16.50
539.681	6.58	864.04	4.42	870.812	22.98	912.397	16.62
542.47	4.83	866.82	6.08	875.547	29.62	984.838	15.96
543.594	3.18	869.77	4.33	865.487	16.89	980.749	15.79
543.594	6.53	875.77	7.84	859.925	31.11	987.158	16.42
540.224	4.81	863.84	6.13	860.421	34.46	983.928	16.11
543.594	3.20	875.80	7.71	865.653	24.67	986.795	16.95
541.864	1.54	873.18	7.82	852.434	26.91	982.305	16.66
532.996	7.28	870.78	6.10	852.788	21.14	987.158	16.34
542.639	3.15	881.49	7.73	875.852	16.54	959.350	16.32
543.594	4.89	872.45	7.70	862.608	22.64	965.887	16.64
542.639	3.21	881.44	7.99	877.978	41.68	988.126	16.76
539.681	3.21	880.16	6.12	844.67	19.51	910.851	16.09
542.47	3.16	867.93	7.85	853.718	29.80	967.817	16.90
542.639	4.85	876.00	7.85	866.724	13.51	970.321	16.45
539.681	3.19	876.96	6.08	860.979	25.35	983.013	15.84
543.594	6.52	877.65	6.12	842.34	17.74	982.437	16.14
532.996	20.57	889.58	7.80	863.448	19.67	968.908	16.81
539.681	4.88	878.53	6.21	874.206	11.48	933.486	16.87
539.681	4.88	879.14	7.81	859.237	25.04	969.703	16.56
543.594	3.18	876.88	6.08	874.501	13.34	955.091	16.14
543.594	3.23	869.34	7.71	878.426	18.49	966.646	16.29
543.594	6.53	874.79	7.81	865.96	37.21	965.034	15.74
532.996	5.78	883.44	6.11	851.286	18.61	955.114	19.47
548.209	3.14	880.38	6.16	879.6	32.45	925.843	19.34
543.594	3.21	883.11	7.75	863.215	28.03	956.963	18.46
539.681	3.26	872.77	9.53	872.659	24.61	983.001	19.00
539.681	4.85	868.74	7.80	854.973	38.40	948.790	20.02

120 customers	time	150 customers	time	199 customers	time
1160.79	14.01	1082.96	31.36	1388.03	40.64
1276.77	24.20	1083.93	28.02	1404.33	43.17
1186.90	25.88	1079.04	27.79	1413.17	41.49
1254.77	24.23	1102.52	26.41	1410.37	65.13
1293.39	24.90	1079.67	19.28	1399.84	39.48
1199.69	24.43	1082.07	18.55	1381.77	41.61
1243.30	25.39	1085.76	17.24	1399.35	49.88
1281.94	24.05	1076.21	17.74	1405.13	54.79
1107.77	30.44	1084.07	18.35	1387.39	53.03
1276.77	24.20	1106.51	19.37	1397.90	56.24
1208.09	24.43	1079.32	19.76	1389.73	53.26
1281.48	25.90	1088.33	19.47	1390.62	54.95
1253.94	24.46	1115.04	18.66	1396.09	57.17
1231.03	27.15	1087.66	19.44	1390.07	49.77
1203.80	25.15	1088.16	18.79	1394.36	59.48
1262.10	25.62	1098.87	19.15	1401.48	52.26
1249.97	26.73	1098.37	17.73	1386.19	54.06
1276.77	24.20	1101.43	17.60	1377.71	58.74
1264.88	26.09	1079.08	19.29	1400.82	59.80
1183.11	25.71	1081.41	17.60	1390.12	52.49
1205.60	26.28	1084.96	17.11	1382.12	58.59
1191.83	25.78	1080.58	17.70	1371.11	54.39
1198.60	32.65	1092.98	18.46	1400.55	52.40
1141.14	29.38	1092.36	18.24	1392.21	58.43
1164.56	30.34	1089.26	19.29	1393.84	53.60
1151.44	29.81	1079.55	17.79	1407.42	59.04
1285.73	29.10	1083.48	17.69	1424.28	63.49
1276.77	24.20	1079.28	19.94	1422.60	68.49
1218.19	25.81	1079.00	21.02	1436.76	48.01
1208.320	26.52	1084.62	27.62	1376.31	54.34

Appendix G

Results of ACO for CVRP

50 customers	time	75 customers	time	100(a) customers	time	100(b) customers	time
543.594	5.9	878.938	8.5	875.908	5.9	957.015	32.4
543.594	6.4	887.819	11.2	874.528	3.5	988.536	15.8
548.534	6.1	885.025	11.0	875.153	6.6	916.123	19.0
543.594	6.4	885.025	11.0	859.619	8.7	923.867	18.2
543.594	6.8	887.031	11.9	884.312	4.1	893.092	27.7
543.594	6.2	888.75	8.0	882.322	3.6	932.294	17.7
543.594	6.9	881.462	21.1	866.957	11.1	895.662	25.7
543.594	6.4	895.308	12.7	876.594	2.8	949.828	26.0
548.534	9.7	896.704	8.4	878.582	2.8	910.01	21.2
543.594	5.6	894.351	9.7	854.631	9.7	943.168	46.9
543.594	10.1	899.975	11.8	864.736	6.3	907.511	16.6
546.383	6.4	888.14	12.4	863.208	5.9	896.092	36.8
543.594	5.8	886.217	12.1	884.1	3.0	851.062	74.6
543.594	5.7	868.882	8.7	882.24	5.1	924.401	18.5
543.594	5.7	886.217	14.7	879.992	8.2	939.059	26.5
543.594	6.7	883.295	6.4	880.83	8.1	920.538	34.0
539.681	7.0	889.203	10.5	888.644	7.3	872.96	26.4
543.594	10.6	882.28	8.9	873.603	9.1	954.133	15.9
543.594	5.9	879.461	8.9	885.346	3.3	890.486	33.3
543.594	7.7	881.987	9.0	875.217	4.7	878.018	26.3
543.594	6.9	888.15	7.3	876.405	4.5	876.112	28.5
543.533	5.6	899.447	8.4	873.042	6.0	898.973	18.5
543.594	5.7	882.756	15.2	906.546	3.4	907.741	49.8
543.594	5.5	886.661	11.6	861.067	10.3	962.909	26.2
539.681	5.7	887.381	8.6	900.326	3.2	937.972	19.1
539.681	7.3	871.48	9.0	871.146	7.0	863.489	29.2
543.594	5.7	887.904	11.8	891.751	8.8	872.69	47.8
543.594	5.9	881.848	12.4	863.671	6.8	927.597	29.9
543.594	5.5	886.971	7.4	880.993	9.0	885.349	14.8
543.594	6.1	871.036	10.2	865.796	13.9	873.41	40.1

120 customers	time	150 customers	time	199 customers	time
1293.49	43.0	1072.66	32.2	1401.23	73.9
1196.29	89.3	1077.89	50.5	1393.02	80.1
1155.17	42.9	1088.46	31.6	1381.66	80.3
1228.15	139.4	1076.89	30.3	1377.91	164.5
1149.48	51.8	1076.89	42.6	1386.61	101.8
1250	69.5	1079.8	39.3	1394.52	115.4
1190.67	68.7	1076.84	51.8	1387.34	169.8
1297.12	14.0	1080.24	49.2	1386.48	132.5
1160.79	40.3	1080.79	39.5	1399.81	41.9
1248.46	25.9	1082.72	30.8	1382.1	112.6
1186.9	24.2	1083.94	30.7	1395.29	84.0
1254.77	24.9	1078.45	33.6	1397.14	54.4
1293.39	24.4	1082.83	25.9	1399.84	46.5
1199.69	25.4	1075.44	47.7	1409.45	53.3
1243.3	24.1	1084.95	25.8	1385.59	112.0
1281.94	24.9	1080.87	33.6	1402.85	52.9
1115.97	24.2	1085.87	58.5	1392.43	60.6
1276.77	24.4	1077.14	31.1	1389.83	85.0
1208.09	25.9	1078.44	32.0	1388.25	104.2
1281.48	24.5	1076.41	32.8	1403.85	54.5
1253.94	27.2	1082.55	28.1	1395.32	69.1
1231.03	25.1	1081.42	38.0	1387.3	136.4
1203.8	25.6	1078.3	69.9	1401.72	73.9
1262.1	26.7	1088.55	33.1	1386.25	51.7
1249.97	24.2	1083.98	34.9	1399.29	71.9
1246.98	26.1	1080.87	24.6	1410	74.7
1264.88	25.7	1068.86	65.7	1388.18	119.6
1183.11	26.3	1077.19	29.7	1384.05	135.7
1205.6	25.8	1085.52	46.5	1399.5	90.0
1191.83	0.0	1076.18	30.2	1397.48	77.5

Appendix H

Results for ant-based hyper-heuristic with TSP heuristics

Oliver30	time	Eil51	time	Eil76	time	KroA100	time
429.425	10.0	435.823	16.3	559.533	56.2	21508.8	112.4
427.899	15.8	445.079	15.0	574.446	46.4	21870.3	111.0
429.425	10.1	443.527	16.1	564.562	52.0	22590.5	112.6
429.425	9.7	435.472	16.4	560.791	53.2	21679.6	98.4
429.425	10.2	441.498	9.1	571.105	35.9	22639.3	103.7
427.899	9.9	441.498	17.2	584.411	45.8	21705.2	103.9
429.425	9.8	435.823	17.3	573.14	35.3	21687.3	111.1
429.425	10.2	443.527	16.8	568.304	54.8	21792.4	100.2
429.425	10.6	441.955	16.4	564.079	51.4	21881	109.8
429.425	10.3	436.089	16.0	567.318	50.3	22583.2	113.3
429.425	10.7	443.527	12.8	567.838	50.5	22288.1	102.0
429.425	9.3	441.955	16.6	570.104	49.5	21998.2	71.9
428.85	9.8	435.823	16.3	568.639	51.9	22054.2	110.1
427.899	10.4	436.336	16.0	573.542	48.7	21839.6	92.4
429.425	10.8	445.058	15.1	567.519	50.2	21458.5	84.7
427.899	15.4	435.823	15.7	585.093	43.7	22007.2	57.9
429.425	10.2	442.526	15.7	559.334	53.8	21482.7	108.2
429.425	9.6	445.264	13.5	566.589	41.1	21756.5	105.3
428.85	9.8	451.579	15.8	567.39	51.3	22170.1	82.4
429.425	10.9	438.78	15.2	567.206	51.5	21879.3	60.4
427.899	17.0	436.604	15.4	562.938	47.1	22327.6	68.2
429.425	11.5	436.109	16.1	576.991	45.9	22266.5	77.0
429.425	9.5	437.355	10.9	563.311	47.7	22439.6	71.1
428.85	14.6	436.109	15.7	581.244	49.5	23183	103.3
428.85	10.8	437.035	15.1	567.206	51.7	21812.4	106.4
429.425	9.8	441.498	15.4	559.334	53.3	22394.5	101.7
428.85	10.9	443.796	16.1	584.411	42.5	21879.3	60.5
429.425	9.8	436.089	15.8	568.304	53.1	21782.3	109.9
429.425	9.9	444.172	15.4	571.1	34.5	22005.4	104.5
429.425	10.5	441.498	15.4	571.105	35.5	22217.4	113.4

Appendix I

Results for ant-based hyper-heuristic with CVRP heuristics

Oliver30	time	Eil51	time	Eil76	time	KroA100	time
429.425	8.2	455.684	13.1	584.451	28.1	22052.2	57.9
429.425	8.2	445.058	13.6	586.41	41.3	21704.7	62.4
429.425	8.3	455.93	9.1	583.246	29.5	22059.6	61.2
429.425	9.5	444.172	11.5	586.589	26.0	22059.6	56.2
429.902	11.2	445.534	13.9	582.902	22.7	21903.5	84.2
429.425	10.0	449.159	11.5	582.573	35.6	22059.6	64.2
427.899	10.3	448.233	15.3	580.828	35.1	22393.7	69.7
429.425	9.0	442.937	8.3	577.447	49.6	21903.5	78.9
429.668	8.3	447.607	7.2	580.062	61.8	22524.2	49.9
429.425	8.2	459.416	11.5	590.852	25.1	21879.3	51.8
429.425	8.4	444.422	9.8	578.423	22.0	21927.8	81.5
429.425	8.2	449.237	9.3	595.003	21.9	22012.4	78.7
428.85	11.6	449.237	8.7	575.682	26.8	21704.7	113.8
428.85	9.0	448.911	8.8	584.35	26.8	21856.3	71.0
429.425	8.7	445.684	15.5	585.301	21.7	21952.5	86.4
429.425	8.0	444.2	17.8	591.166	23.2	22067.7	53.1
429.425	8.5	442.264	17.8	585.033	31.3	21991.18	53.7
429.425	9.4	448.615	18.0	588.261	25.1	22115.3	95.5
429.425	8.7	443.217	18.3	569.119	56.7	22271.4	62.2
429.425	8.6	443.546	18.5	587.62	37.3	22226.5	54.8
428.85	8.0	447.975	9.7	595.003	21.2	21911.6	110.8
429.425	9.2	449.594	7.8	585.285	44.1	21905.3	54.4
427.899	10.6	462.386	6.9	593.422	21.3	22271.4	97.0
429.425	8.7	447.607	11.4	578.716	45.5	22562.5	47.0
429.425	8.3	448.372	15.4	586.661	38.4	22123.4	114.1
428.85	9.2	446.345	8.8	582.999	61.2	22593.3	103.1
429.425	8.7	444.9	14.6	583.536	43.1	22162.5	58.4
429.425	9.3	434.846	18.0	587.052	36.6	21856.3	98.9
429.425	8.0	446.688	8.0	588.655	21.4	22016.6	87.9
429.425	8.2	437.621	17.1	591.661	54.0	22059.6	70.1

Appendix J

Results for ACO hyper-heuristic with TSP heuristics

Oliver30	time	Eil51	time	Eil76	time	KroA100	time
429.902	8.6	434.026	32.0	575.866	239.9	22059.6	64.2
428.85	10.4	442.441	60.9	583.976	177.9	22393.7	69.7
429.425	9.7	444.125	64.2	586.185	133.9	21879.3	51.8
429.425	9.6	439.374	35.3	581.368	186.7	22067.7	53.1
429.425	7.1	434.982	51.5	572.128	238.9	22052.2	57.9
429.902	7.4	436.093	39.4	594.258	86.2	21903.5	84.2
429.425	9.3	436.278	81.5	584.093	158.7	21704.7	62.4
429.902	8.0	434.989	63.8	586.356	108.3	21856.3	71.0
427.899	9.2	431.387	87.0	580.076	158.9	22059.6	70.1
429.425	8.1	438.567	46.7	594.177	83.1	22059.6	61.2
429.425	9.4	435.206	90.0	581.212	148.9	22271.4	62.2
429.425	9.3	430.811	72.2	584.093	112.2	21991.8	53.7
429.425	8.5	458.629	34.2	570.248	240.6	22115.3	95.5
429.425	10.7	434.069	32.0	590.485	84.3	22226.5	54.8
429.902	13.5	435.792	25.0	584.451	105.2	21911.6	110.8
429.902	9.0	429.699	63.1	568.892	260.6	21905.3	54.4
429.902	7.5	443.382	44.9	583.714	128.7	22123.4	114.1
429.902	9.1	429.699	63.1	587.514	81.4	21704.7	113.8
429.425	8.5	463.451	25.1	586.185	163.4	21927.8	81.5
429.425	7.7	434.069	35.6	575.866	239.9	21903.5	78.9
429.425	9.7	439.076	72.9	586.185	117.2	21952.5	92.2
429.425	11.1	435.792	24.8	575.866	223.2	22059.6	56.2
429.425	11.1	430.811	72.2	586.185	165.1	22016.6	87.9
428.85	7.6	435.279	25.8	581.368	153.4	22539.3	103.1
429.425	9.0	435.206	90.9	586.185	150.5	22012.4	78.7
429.902	12.8	439.076	73.1	586.185	150.5	22562.5	47.0
428.376	12.2	463.451	25.1	586.185	145.5	22271.4	97.0
429.902	9.1	439.076	72.9	575.866	242.6	21856.3	98.9
429.902	9.2	434.069	35.6	581.368	168.4	22162.5	58.4
429.425	8.5	463.451	25.1	575.866	173.2	22524.2	49.9

References

Aarts E., Korst J. & Michiels W. (2005). Simulated Annealing. In Burke E.K. & Kendall G. (eds), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, (pp. 187-210). Springer.

Abdullah S. & Burke E. K. (2006). A Multi-start Large Neighbourhood Search Approach with Local Search Methods for Examination Timetabling. In Long D., Smith S. F., Borrajo D., and McCluskey L., (eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2006) (pp. 334-337).

Aickelin U. (1998). Nurse Rostering with Genetic Algorithms. In Proceedings of Young Operational Research Conference (12). University of Surrey, Guilsford.

Alexandrov D.A.& Kochetov Y.A. (1999). The Behaviour of the Ant Colony Algorithm for the Set Covering Problem (pp. 255-260). Operation Research Proceedings.

Applegate D., Bixby R., Chvatal V. & Cook W. (1998). On the Solution of Traveling Salesman Problems. In Mathematica (Extra Volume ICM, Chapter 3, pp. 645-656).

Asmuni H., Burke E. K., & Garibaldi J. M. (2004). Fuzzy Multiple Ordering Criteria for Examination Timetabling. In Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT), (pp. 51-66).

Ayob M. (2005). Optimisation of Surface Mount Device Placement Machine in Printed Circuit Board Assembly. PhD Thesis, School of Computer Science and Information Technology, University of Nottingham.

Ayob M. & Kendall G. (2003). A Monte Carlo Hyper-heuristic to Optimise Component Placement Sequencing for Multi Head Placement Machine. In Proceedings of the International Conference on Intelligent Technologies (InTech'03, pp. 132-141).

Bai R. (2005). An Investigation of Novel Approaches for Optimizing Retail Shelf Space Allocation. PhD Thesis, School of Computer Science and Information Technology, University of Nottingham.

Bai R. & Kendall G. (2005). An Investigation of Automated Planograms using a Simulated Annealing Based Hyper-heuristics. In Ibaraki, T., Nonobe, K., & Yagiura, M. (Eds.), Metaheuristics: Progress as Real Problem Solvers. Operations Research/Computer Science Interfaces, (Vol. 32, pp. 87-108). Springer: Berlin, Heidelberg, New York.

Bai R., Blazewicz J., Burke E. K., Kendall G. & McCollum B. (2007). A Simulated Annealing Hyper-Heuristic Methodology for Flexible Decision Support. In Technical Report NOTTCS-TR-2007-8. School of Computer Science, University of Nottingham.

Baker B.M. & Sheasby J. (1999). Extensions to the Generalized Assignment for Vehicle Routing. In European Journal of Operation Research., (volume 119, pp. 147-157).

Bell J.E. & McMullen P.R. (2004). Ant Colony Optimization Techniques for the Vehicle Routing Problem. In Advanced Engineering Informatics, (volume 18, pp. 41-48).

Bentley J.L. (1992). Fast Algorithms for Geometric Traveling Salesman Problems. In ORSA Journal of Computing (volume 4(4), pp. 387 411).

Bin Y., Zhang-Zhen Y. & Yao B. (2009). An Improved Ant Colony Optimization for Vehicle Routing Problem. In European Journal of Operation Research, (volume 196, pp. 171-176).

Blum C. & Roli A. (2003). Metaheuristics in Combinatorial Optimisation: Overview and Conceptual Comparison. In ACM Computing Surveys, (volume 35(3), pp. 268-308).

Braysy O. & Gendreau M. (2005a). Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. In Transportation Science, (volume 39, pp. 104-118).

Braysy O. & Gendreau M. (2005b). Vehicle Routing Problem with Time Windows, Part II: Metaheuristics, In Transportation Science (volume 39, pp. 119-139).

Braysy O., Dullaert W. & Gendreau M. (2004). Evolutionary Algorithms for the Vehicle Routing Problem with Time Windows. In Journal of Heuristics, (volume 10, pp. 587-561).

Bullnheimer B., Hartl R.F. & Strauss C. (1999a). Applying the Ant System to the Vehicle Routing Problem, In MetaHeuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic.

Bullnheimer B., Hartl R. F. & Strauss C. (1999b). A New Rank Based Version of the Ant System - A Computational Study. In Central European Journal of Operations Research, (volume 7, pp. 25-38).

Bullnheimer B., Hartl R.F. & Strauss C. (1999c). An Improved Ant System Algorithm for the Vehicle Routing Problem. In Annals of Operations Research, (volume 89, pp. 319–328).

Burke E. K. & Soubeiga E. (2003). Scheduling Nurses using a Tabu-Search Hyperheuristic. In Proceedings of the 1st Multidisciplinary International. Conference. on Scheduling: Theory and Applications (MISTA 2003), (volume 1, pp. 197-218).

Burke E. K., De Causmaecker P. & Vanden Berghe G. (1999). A Hybrid Tabu Search Algorithm for the Nurse Rostering Problem. In Lecture Notes in Artificial Intelligence, Springer, (volume 1585, pp. 187-194).

Burke E. K., De Causmaecker P., Petrovic S. & Vanden B.G. (2004). Variable Neighborhood Search for Nurse Rostering Problems. In Resende M.G.C. and De Souza J.P. (eds.), Metaheuristics: Computer Decision Making, Combinatorial Optimization, Kluwer, (pp. 153-172).

Burke E. K., Dror M., Petrovic S. & Qu R. (2005a). Hybrid Graph Heuristics Within a Hyper-heuristic Approach to Exam Timetabling Problems. In Golden B. L., Raghavan S., and Wasil E. A., (eds.), The Next Wave in Computing, Optimization, and Decision Technologies Conference Volume of the 9th Informs Computing Society Conference, Springer, (pp. 79-91).

Burke E. K., Kendall G., Landa-Silva J. D., O'Brien R. & Soubeiga E. (2005b). An Ant Algorithm Hyperheuristic for the Project Presentation Scheduling Problem. In Proceedings of the 2005 IEEE Congress on Evolutionary Computation, (volume 3, pp.2263-2270).

Burke E.K. & Kendall G. (2005). Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques (pp. 5-18). Springer.

Burke E.K., Hyde M., Kendall G., Ochoa G., Ozcan E. & Qu R. (2010a). Hyper-heuristics: A Survey of the State of the Art. In Technical Report No. NOTTCS-TR-SUB-0906241418-2747. School of Computer Science and Information Technology, University of Nottingham, Computer Science.

Burke E.K., Kendall G. & Soubeiga E. (2003a). A Tabu-search Hyperheuristic for Timetabling and Rostering. In Journal of Heuristics, (volume 9(6), pp. 451-470).

Burke E.K., Kendall G., Newall J., Hart E. & Ross P. (2003b). Hyper-heuristics: An Emerging Direction in Modern Search Technology. In Glover F. & Kochenberger G.A. (eds), Handbook of Metaheuristics (pp. 457-474). Kluwer Academic Publishers.

Burke E.K., Kendall G., O'Brien R.F.J., Redrup D. & Soubeiga E. (2003c). An Ant Algorithm Hyper-heuristic. In Proceedings of the Fifth Meta-heuristics International Conference (MIC2003).

Burke E.K., Landa-Silva J.D. & Soubeiga E. (2005c). Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling. In Meta-heuristics: Progress as Real Problem Solvers, (pp. 129-158).

Burke E.K., Hyde M., Kendall G., Ochoa G., Ozcan E., & Woodward J. (2010b). A Classification of Hyper-heuristics Approaches, In Gendreau M. & Potvin J-Y. (eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, (pp. 449-468). Springer.

Chambers J.B. & Barnes J.W. (1996). Tabu Search for the Flexible-routing Job Shop Problem. In Technical Report Series ORP 96-10. Department of Mechanical Engineering, University of Texas at Austin.

Chen P. (2006). Hyper-heuristic Ant Algorithm for the Traveling Tournament Problem. PhD Thesis. University of Nottingham, UK.

Chen P., Kendall G. & Berghe G.V. (2007). An Ant Based Hyper-heuristic for the Travelling Tournament Problem. In Proceedings of IEEE Symposium of Computational Intelligence in Scheduling (CISched 2007), (pp. 19-26).

Christofides N. & Eilon S. (1969). An Algorithm for the Vehicle Dispatching Problem. In Operation. Research. Quart, (volume 20, pp. 309-318).

Clarke G. & Wright J.W. (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. In Operations Research, (volume 12, pp. 568-581).

Colorni A., Dorigo M. & Maniezzo V. (1992). An Investigation of Some Properties of An Ant Algorithm. In Proceedings of PPSN-II, Second International Conference on Parallel Problem Solving from Nature Conference, (pp. 509-520). Elsevier, Amsterdam.

Connolly D.T. (1990). An Improved Annealing Scheme for the QAP. In European Journal of Operational Research, (volume 46, pp. 93-100).

Cook S.A. (1971). The Complexity of Theorem-proving Procedures. In Proceedings of 3rd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery (pp. 151-158). New York.

Cordeau J.F., Gendreau M, Laporte G., Potvin J-Y. and Semet F. (2002). A Guide to Vehicle Routing Heuristics. In The Journal of Operational Research Society, (volume 53(5), pp. 512-522).

Cordeau J-F. & Laporte G. (2005). Tabu Search Heuristics for Vehicle Routing Problem. In Metaheuristic Optimization via Memory and Evolution, (volume 30, pp. 145-163).

Costa D. & Hertz A. (1997). Ants can colour graphs. In The Journal of the Operational Research Society, (volume 48, pp. 295-305).

Cowling P., Kendall G. & Soubeiga E. (2000). A Hyperheuristic Approach to Scheduling a Sales Summit. In Burke E. K. & Erben W. (eds.), Practice and Theory of Automated

Timetabling III: Third International Conference (PATAT 2000), Lecture Notes in Computer Science 2079, (pp. 176-190). Springer-Verlag.

Cowling P., Kendall G. & Soubeiga E. (2002a). Hyperheuristics: A Robust Optimisation Method Applied to Nurse Scheduling. In Seventh International Conference on Parallel Problem Solving from Nature, PPSN, (pp. 851-860). LCNS Springer.

Cowling P., Kendall G., & Han L. (2002b). An Adaptive Length Chromosome Hyperheuristic Genetic Algorithm for a Trainer Scheduling Problem. In Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02), (pp. 267-271).

Cowling P., Kendall G., & Soubeiga E. (2002b). Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation. In Cagoni S., Gottlieb J., Hart E., Middendorf M., & Goenther R.(eds.), Applications of Evolutionary Computing: Proceeding of Evo Workshops, Lecture Notes in Computer Science,(volume 2279, pp. 1-10). Springer-Verlag.

Cowling P., Kendall G. & Han, L. (2002d). An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. In Proceedings of the Congress on Evolutionary Computation 2002 (pp. 1185-1190), CEC 2002. Morgan Kaufman.

Cowling P., Kendall G., & Soubeiga E. (2001a). A Hyperheuristic Approach to Scheduling a Sales Summit. In selected papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT 2001), Springer.

Cowling P., Kendall G., & Soubeiga E., (2001b). A Parameter-free Hyperheuristic for Scheduling a Sales Summit. In Proceedings of the 4th Metaheuristic International Conference (MIC 2001), (pp. 127-131).

Crowston W.B., Glover F., Thompson G.L. & Trawick J.D. (1963). Probabilistic and Parametric Learning Combinations of Local Job Shop Scheduling Rules. In ONR research memorandum. Cernegie-Mellon University Pittsburgh.

Dantzig G.B., Fulkerson R. & Johnson S.M. (1954). Solution of a large-scale Traveling Salesman Problem. In Operations Research 2, (pp. 393-410).

Dantzig G.B. & Ramser J.H. (1959). The Truck Dispatching Problem. In Management Science, (volume 6(1), pp. 80-91).

Den Besten M., Stutzle T. & Dorigo M. (2000). Ant Colony Optimization for the Total Weighted Tardiness Problem. In Proceedings of the Parallel Problem Solving from Nature Conference, (pp. 611-620). Springer Verlag.

Deneubourg L., Aron S., Goss S. & Pasteels J.M. (1990). The Self Organizing Exploratory Pattern of the Argentine Ant. In Insect Behaviour, (volume 3, pp. 159-168).

Dethloff J. (2001). Vehicle Routing and Reverse Logistics: The Vehicle Routing Problem with Simultaneous Delivery and Pick-up. In OR Spectrum, (volume 23, pp. 79-96).

Di Caro G. & Dorigo M. (1997). AntNet: A Mobile Agents Approach to Adaptive Routing. In Technical Report IRIDIA/97-12. Universite Libre de Bruxelles, Belgium.

Di Caro G. & Dorigo M. (1998). AntNet: Distributed Stigmergetic Control for Communications Networks. In Journal of Artifcial Intelligence Research, (volume 9, pp. 317-365).

Donati A.V., Montemanni R., Casagrande N., Rizzoli A.E. & Gambardella L.M. (2003). Time Dependent Vehicle Routing Problem with a Multi Ant Colony System. In Technical Report TR-17-03. Idsia, Galleria 2, Manno, 6928, Switzerland.

Dorigo M. & Di Caro G. (1999a). Ant Algorithms for Discrete Optimization. Artificial Life, (volume 5, pp. 137-172).

Dorigo M. & Di Caro G. (1999b). The Ant Colony Optimization Meta-heuristic. New Ideas in Optimization (pp. 11-32). Mc Graw Hill.

Dorigo M. & Di Caro G. (1999c). Ant Colony Optimization: A New Meta-heuristic. In Proceedings of the Congress on Evolutionary Computation, (pp. 1470-1477). IEEE Press.

Dorigo M. & Gambardella L. M. (1997). Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. In IEEE Transaction on Evolutionary Computation, (volume 1, pp. 53-66).

Dorigo M. & Gambardella L.M. (1995). Ant-Q - A Reinforce Learning Approach to the Traveling Salesman Problem. In Proceedings of ML-95, Twelve International Conference on Machine Learning, (pp. 252-260). Morgan Kaufmann.

Dorigo M. & Stutzle T. (2004). Ant Colony Optimization. MIT Press.

Dorigo M. & Stutzle T. (2003a). Handbook of Metaheuristics, Glover F. & Kochenberger G.A. (eds) (pp. 251-285). Kluwer Academic Publishers.

Dorigo M. & Stutzle T. (2003b). The Ant Colony Optimization Metaheuristic: Algorithms, Applications and Advances. Handbook of Metaheuristics, Kluwer Academic Publishers, (pp. 251-285).

Dorigo M. & Stutzle T. (2004). From Real to Artificial Ants. Ant Colony Optimization (pp. 1-23). The MIT Press.

Dorigo M., Maniezzo V. & Colorni A. (1991). Positive feedback as a Search Strategy. In Technical Report No 91-016. Dipartimento di Eletronica, Politecnico di Milano, Italy.

Dorigo M., Maniezzo V. & Colorni A. (1996). Ant system: Optimisation by a Colony of Cooperating Agents. In IEEE Transactions on Systems, Man and Cybernetics, (pp. 26, 29-41).

Dorigo, M. & Gambardella L.M. (1997). Ant Colonies for the Travelling Salesman Problem. In Biosystems, (volume 43, pp. 73-81).

Dowsland K. A., Soubeiga E. & Burke E. K. (2007). A Simulated Annealing Hyper-heuristic for Determining Shipper Sizes. In European Journal of Operational Research, (volume 179(3), pp. 759-774).

Fang H., Ross P. & Corne D. (1993). A Promising Genetic Algorithm Approach to Job Shop Scheduling, Rescheduling and Open-shop Scheduling problems. In Forrest S. (ed), Fifth International Conference on Genetic Algorithm (pp. 375-382), Morgan Kaufmann, San Mateo.

Feo T.A. & Resende M.G.C. (1989). A Probabilistic for a Computationally Difficult Set Covering Problem. In Operation Research Letters, (volume 8, pp. 67-71).

Fisher H & Thompson G.L. (1961). Probabilistic Learning Combinations of Local Job-shop Scheduling Rules. In Factory Scheduling Conference. Carnegie Institute of Technology.

Fisher H & Thompson G.L. (1963). Probabilistic Learning Combinations of Local Job-shop Scheduling Rules. In J.F. Muth & G.L. Thompson (eds), Industrial Scheduling, (pp. 225-251). Prentice-Hall, Inc, New Jersey.

Fisher M.L. & Jaikumar R. (1981). A Generalized Assignment Heuristic for the Vehicle Routing Problem. Networks, (volume 11, pp. 109-124).

Fraser A.S. (1957). Simulation of Genetic Systems by Automatic Digital Computers. In Australia Journal of Biological Sciences, (volume 10, pp. 484-491).

Gambardella L.M., Taillard E. & Dorigo M. (1999). Ant Colonies for the Quadratic Assignment Problem. In Journal of Operational Research Society, (volume 50, pp. 167–176).

Garey M.R. & Johnson D.S. (1979). Computers and Intractabiliy: A Guide to the Theory of NP-Completeness, W.H. Freeman.

Garrido P. & Castro C. (2009). Stable solving of CVRPs using Hyperheuristics. In Genetic and Evolutionary Computation Conference (GECCO'09) (pp. 255-262). ACM, Montreal, Canada.

Gaspero L.D & Schaerf A. (2007). A Composite-neighborhood Tabu Search Approach to the Travelling Tournament Problem. In Journal of Heuristics, (volume 13(2), pp. 189-207).

Gaspero L.D. & Schaerf A. (2001). Tabu Search Techniques for Examination Timetabling. In Lecture Notes in Computer Science, (volume 2079, pp. 104-117).

Gendreau M. & Potvin J. (2005a). Tabu Search. In Burke E.K. and Kendall G. (eds), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, (pp. 165-186). Springer.

Gendreau M. & Potvin J. (2005b). Metaheuristics in Combinatorial Optimization. In Annal of Operations Research, (volume 140, pp. 189-213).

Gendreau M. & Potvin J-Y. (1998). Dynamic Vehicle Routing and Dispatching. In Crainic T.G. and Laporte G. (eds), Fleet Management and Logistic, (pp. 115-226).

Gendreau M., Hertz A. & Laporte G. (1994). A Tabu Search Heuristic for the Vehicle Routing Problem. In Management Science, (volume 40(10), pp. 1276–1290).

Gillet B.E. & Miller L.R. (1974). A Heuristic Algorithm for the Vehicle Dispatch Problem. In Operations Research, (volume 22, pp. 340-349).

Glover F & Kochenberger G.A. (2003). Hyper-heuristics: An Emerging Direction in Modern Search Technology. Handbook of Metaheuristics, Kluwer Academic Publishers.

Glover F. & Laguna M. (1997). Tabu Search. Kluwer Academic Publisher.

Gratch J. & Chien S. (1996). Adaptive Problem-solving for Large-scale Scheduling Problems: a Case Study. In Journal of Artificial Intelligence Research. (volume 4, pp. 365-396).

Gratch J., Chien S. & DeJong G. (1993). Learning Search Control Knowledge for Deep Space Network Scheduling. In Proceedings of the Tenth International Conference on Machine Learning (pp. 135-142). Amherst, MA.

Gutin G. & Punnen A.P. (2002). The Traveling Salesman and Its Variations. Springer.

Han L. & Kendall G. (2003b). Investigation of a Tabu Assisted Hyper-heuristic Genetic Algorithm. In Proceedings of the Congress on Evolutionary Computation (CEC2003), (volume 3, pp. 2230-2237).

Han L., Kendall G & Cowling. (2002). An Adaptive Length Chromosome Hyperheuristics Genetic Algorithm for a Trainer Scheduling Problem. In Proceedings of the 4th Asia Pasific Conference on Simulated Evolution and Learning, (pp. 267-271). SEAL02, Singapore.

Hansen P. & Mladenovic N. (2001). Variable Neighborhood Search: Principles and Applications. In European Journal of Operational Research, (volume 130, pp. 449-467).

Hansen P. & Mladenovic N. (2003). Variable Neighborhood Search. In Glover F. & Kochenberger G.A. (eds), Handbook of Metaheuristics (pp 145-184). Kluwer Academic Publishers.

Hansen P. & Mladenovic N. (2005). Variable Neighborhood Search. In Burke E.K. & Kendall G. (eds), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques (pp. 211- 238). Springer.

Hart E. & Ross P. (1998). A Heuristic Combination Method for Solving Job-shop Scheduling Problems. Parallel Problem Solving from Nature, In Eiben AE., Back T., Schoenauer M. & Schwefel HP. (eds), Lecture Notes in Computer Science, (volume 1498, pp.845-854). Springer-Verlag.

Helsgaun K. (2000). An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. In European Journal of Operational Research, (volume 126, pp. 106-130).

Henderson D. Jacobson S.H. & Johnson A.W. (2003). The Theory of Simulated Annealing. In Glover F. & Kochenberger G.A. (eds), Handbook of Metaheuristics, (pp.287-319). Kluwer Academic Publishers.

Hertz A., Taillard E. & De Werra D. (1995). A Tutorial on Tabu Search. In Proceeding of Giornate di Lavoro AIRO'95, (Entreprise Systems: Management of Technological and Organizational Changes), (pp. 13-24).

Holland J.H. (1992). Genetic Algorithms. Scientific American, July, (volume 267(1), pp. 66-72).

Hussin N. M. (2005). Tabu Search Based Hyper-heuristic Approaches for Examination Timetabling. PhD thesis. School of Computer Science and Information Technology, University of Nottingham.

Ichoua S., Gendreau M. & Potvin J-Y. (2003). Vehicle Dispatching with Time-dependent Travel Times. In European Journal of Operational Research, (volume 144(2), pp. 379-396).

Karp R.M. (1972). Reducibility Among Combinatorial Problems. Plenum Press, (pp. 85-103).

Kendall G & Mohamad M. (2004a). Channel Assignment in Cellular Communication Using Great Deluge Hyper-heuristic. In Proceeding of the 2004 IEEE International Conference on Network (ICON2004) (pp. 769-773), Singapore.

Kendall G & Mohamad M. (2004b). Channel Assignment Optimization Using Hyper-heuristic. In Proceeding of the 2004 IEEE Conference on Cybernetic and Intelligent Systems (CIS2004) (pp. 780-795), Singapore.

Kendall G. & Mohd Hussin N. (2004a). An Investigation of a Tabu Search Based Hyper- for Examination Timetabling. In Kendall G., Burke E. K., & Petrovic S. (eds.) Selected Papers from MISTA 2003. Kluwer Publication.

Kendall G. & Mohd Hussin N. (2004b). Tabu Search Hyper-heuristic Approach to the Examination Timetabling Problem at University Technology MARA. In Burke E. K. & Trick M. (eds.) Proceedings of the 5th international Conference on the Practice and Theory of Automated Timetabling (PATAT), (pp. 199-217).

Kendall G., Cowling P. & Soubeiga E. (2002). Choice Function and Random Hyperheuristics. In Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning, (pp. 667-671).

Kirkpatrick, S. Gelatt, C.D. & Vecchi, M.P. (1983). Optimization by Simulated Annealing, In Science, (volume 220, pp. 671-380).

Koza JR. & Poli R. (2005). Genetic Programming. In Burke E.K. & Kendall G. (eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, (pp.127-164). Kluwer, Boston.

Lawer, E.L. (1976). Combinatorial Optimization: In Networks and Matroids. New York: Holt, Rinehart and Winston.

Laporte G. (1992). The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms. In European Journal of Operational Research, (volume 59, pp. 345-358).

Laporte G. (2009). Fifty Years of Vehicle Routing. In Transportation Science, Articles in Advance, (pp. 1-9).

Laporte G. & Semet F. (2002). Classical Heuristics for the Capacitated VRP. In Toth P. and Vigo D. (eds), SIAM Monographs on Discrete Mathematics and Applications, Chapter 5, (volume 9, pp. 109-128).

Lim A., Rodrigues B. & Zhang X. (2006). A Simulated Annealing and Hill-climbing Algorithm for the Traveling Tournament Problem. In European Journal of Operational Research. (volume 174(3), pp. 1459-1478).

Lin S. & Kernighan B.W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. In Operations Research, (volume 21, pp. 98-516).

Lysgaard J., Letchford N. & Eglese R.W. (2004). A New Branch-and-Cut Algorithm for Capacitated Vehicle Routing Problem. In Mathematical Programming, Ser. A, (volume 100: pp.423-445).

Maniezzo V. & Colorni A. (1994). The Ant System Applied to the Quadratic Assignment Problem. In Technical Report 94/28. IRIDIA, Université Libre de Bruxelles, Bruxelles, Belgium.

Martins S.L., Pardalos P.M., Resende M. G.C. & Ribeiro C. (1994). Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. In DIMACS Series on Discrete Mathematics and Theoretical Computer Science, (volume 16, pp.237-261). American Mathematical Society.

Michalewicz Z. & Fogel D.B. (2004). How to Solve it: Modern Heuristics. Springer.

Naddef D. & Rinaldi G. (2002). Branch and Cut Algorithms for the Capacitated VRP. In Toth P. and Vigo D. (eds), The Vehicle Routing Problem (SIAM), (pp. 53-84).

O'Brien R.F.J. (2007). Ant Algorithm Hyperheuristic Approaches for Scheduling Problems. PhD Thesis, University of Nottingham. UK.

Ochoa G., Vazquez-Rodriguez JA., Petrovic S. & Burke E. (2009). Dispatching Rules for Production Scheduling: A hyper-heuristic Landscape Analysis. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009), (pp. 1873-1880).

Osman I.H. (1993). Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem. In Annals of Operations Research, (volume 41(4), pp. 421-451). Springer Netherlands.

Osman I.H. (2000). Meta-heuristics: A General Framework. In Proceedings of the International Workshop on Algorithm Engineering, (pp. 117-118). Research Institute for Mathematical Science, Kyoto University, Japan.

Osman I.H. & Kelly J.P. (1996). Meta-heuristics: Theory and Applications. Dordrecht, Netherlands, Kluwer.

Pardalos P.M., Migdalas A. & Burkard R.E. (2002). Combinatorial and Global Optimization. World Scientific.

Pisinger D & Ropke S. (2007). A General Heuristic for Vehicle Routing Problems. In Computers and Operations Research, (volume 34, pp. 2403-2435).

Qu R. & Burke E.K. (2005). Hybrid Variable Hyperheuristics for Exam Timetabling Problems. In The Sixth Metaheuristics International Conference, (pp. 22-26).

Ralphs T.K., Kopman L., Pulleyblank W.R. & Trotter L.E. (2003). On the Capacitated Vehicle Routing Problem. Math Program, (volume 94, pp. 343-359).

Reeves C. (1995). Modern Heuristics Techniques for Combinatorial Problems. McGraw Hill.

Reeves C. (2003). Genetic Algorithms. In Glover F. and Kochenberger G.A. (eds), Handbook of Metaheuristics, (pp. 55-82). Kluwer Academic Publishers.

Resende M.G.C. (1995). Greedy Randomized Adaptive Search Procedures. In Journal of Global Optimization, (volume 6, pp. 109–133).

Resende M.G.C. & Ribeiro C.C. (2003). Greedy Randomized Adaptive Procedures. In Glover F & Kochenberger (eds), Handbook of Metaheuristics (pp.219-249). Kluwer Academic Publishers.

Ross P., Hart E. & Corne D. (1998). Some Observations about GA-based Exam Timetabling. In Burke E.K. & Carter M. (eds), Practice and Theory of Automated Timetabling II, Lecture Notes in Computer Science, (volume 1408, pp. 115-129). Springer-Verlag.

Ross P., Marin-Blazquez JG. & Hart E. (2004). Hyper-heuristics Applied to Class and Exam Timetabling Problems. In Proceeding of the 2004 IEEE Congress on Evolutionary Computation, (pp. 1691-1698). IEEE Press, Portland, Oregon.

Sastry K., Goldberg D. & Kendall G. (2005). Genetic Algorithms. In Burke E.K. & Kendall G. (eds), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, (pp. 97-125). Springer.

Skorin-Kapov J. (1990). Tabu Search applied to the Quadratic Assignment Problem. In ORSA Journal on Computing, (volume 2(1), pp. 33-45).

Soubeiga E. (2003). Development and Application of Hyperheuristics to Personnel Scheduling. PhD thesis. School of Computer Science and Information Technology, University of Nottingham.

Stutzle T, & Hoos H.H. (2000). *MAX-MIN* Ant System. In Journal of Future Generation Computer Systems, (volume 16, pp. 889–914).

Stutzle T. & Dorigo M. (1999). ACO Algorithms for the Quadratic Assignment Problem. New Ideas in Optimization (pp. 33-50), McGraw Hill.

Suman B & Kumar P. (2006). A Survey of Simulated Annealing as a Tool for Single and Multiobjective Optimization. In Journal of Operational Research Society, (volume 57(10), pp. 1143-1160).

Suwannarongsri S. & Puangdownreong D. (2012). Adaptive Tabu Search for Traveling Salesman Problems. In International Journal of Mathematics and Computer in Simulation, (volume 6(2)).

T.H. & Wonnacott R.J. (1990). Introduction Statistics for Business and Economics, (pp. 196-203). John Wiley.

Taillard E.D. (1993). Parallel Iterative Search Methods for Vehicle Routing Problem. In Networks, (volume 23, pp. 661-673).

Terashima-Marin H., Zarate CJF., Ross P. And Valenzuela-Rendon M. (2006). A GA-based Method to Produce Generalized Hyper-heuristics for the 2D-regular Cutting Stock Problem. In Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO 06 (pp. 591-598). ACM Press, New York, USA.

The Traveling Salesman Library. (2008). Website: http://www.tsp.gatech.edu/

Vazquez-Rodriquez JA., Petrovic S. & Salhi A. (2007). A Combined Meta-heuristics with Hyper-heuristic Approach to the Scheduling of the Hybrid Flow Shop with Sequence Dependent Setup Times and Uniform Machines. In Proceedings of the 3rd Multiciplinary International Scheduling Conference: Theory and Applications (MISTA 2007).

Vehicle Routing Datasets. (2003). Website: http://www.coin-or.org/SYMPHONY/branchandcut/VRP/data.

Voss S. (2001). Meta-heuristics: The State of the Art. In Lecture Notes in Computer Science, (volume 2148, pp. 1-23). Springer Berlin/Heidelberg.

Whitley D., Starkweather T. & Fuquay D. (1989). Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. In J. Schaffer (ed), Proceedings of the Third International Conference on Genetic Algorithms Morgan Kaufmann (pp. 133-140), Los Altos, CA.