Design of Vehicle Routing Problem Domains for a Hyper-Heuristic Framework

James Walker

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

December 2015

Abstract

The branch of algorithms that uses adaptive methods to select or tune heuristics, known as hyper-heuristics, is one that has seen a large amount of interest and development in recent years. With an aim to develop techniques that can deliver results on multiple problem domains and multiple instances, this work is getting ever closer to mirroring the complex situations that arise in the corporate world. However, the capability of a hyper-heuristic is closely tied to the representation of the problem it is trying to solve and the tools that are available to do so.

This thesis considers the design of such problem domains for hyper-heuristics. In particular, this work proposes that through the provision of high-quality data and tools to a hyper-heuristic, improved results can be achieved. A definition is given which describes the components of a problem domain for hyper-heuristics. Building on this definition, a domain for the Vehicle Routing Problem with Time Windows is presented. Through this domain, examples are given of how a hyper-heuristic can be provided extra information with which to make intelligent search decisions. One of these pieces of information is a measure of distance between solution which, when used to aid selection of mutation heuristics, is shown to improve results of an Iterative Local Search hyper-heuristic. A further example of the advantages of providing extra information is given in the form of the provision of a set of tools for the Vehicle Routing Problem domain to promote and measure 'fairness' between routes. By offering these extra features at a domain level, it is shown how a hyper-heuristic can drive toward a fairer solution while maintaining a high level of performance.

Acknowledgements

I would like to thank my supervisor, Andrew Parkes, and my previous supervisors Edmund Burke, Gabriela Ochoa, Matthew Hyde and Sanja Petrovic for their support and assistance during the study for this thesis. I would also like to thank Michel Gendreau for his advice regarding the vehicle routing problem and the intial HyFlex domain.

I would like to thank my wife, Lizi, for her love and support throughout my studies.
I would like to express my gratitude to the EPSRC for providing the funding for this work.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

For many years within optimisation research the development and fine-tuning of a variety of methods and algorithms has led to increasingly stronger results over a variety of hard computational search problems. Often, though, these algorithms have been manually tuned specifically to work well on one problem or even on one instance of a single problem. The consequences of this are a range of algorithms that work extremely well on particular problem structures but lack the ability to react well to changes without significant human input. These changes are an integral aspect of many industrial applications, where shifts can change, machinery can break down and traffic can delay deliveries, all at very short notice. Within recent years, study has increased on a class of algorithms, known as hyper-heuristics, that have the potential to adapt to such circumstances with greater ease than has previously been demonstrated. A hyper-heuristic (simply put, a heuristic which selects or creates other heuristics[31]) can overcome a lack of knowledge of problem-specific information to deliver solutions that are of a sufficient quality, without the need for lengthy run-times or excessive human input. These 'good enough - soon enough - cheap enough' [22] solutions are often of far more practical value than a solution that might be of marginally better quality, but a quality that is achieved at a far higher cost.

In order to analyse their performance, hyper-heuristics need representations of combinatorial optimisation problems to be tested on. The relationship between these problem domains and the hyper-heuristics which operate on them is complex. The domains must offer a hyper-heuristic the tools it needs to navigate the search space and improve solution but at the same time may attempt to shield the hyper-heuristic from problem-specific information. An essential question when designing a problem domain is how much information is appropriate, and useful, to offer to a hyper-heuristic. This question is the primary motivation behind the work in this thesis. The issue of designing problem domains for hyper-heuristics will be explored in detail. It will be explored which pieces of information can benefit a hyper-heuristic, as well as how the hyper-heuristic must operate in order to unlock these benefits. This work will also consider how problem domain design might be tackled to represent some ’real-world’ features, and how a hyper-heuristic can interact with these features.

1.2 Contributions of thesis

This thesis will contribute to knowledge and understanding in the following ways.

- A definition will be provided for a problem domain for hyper-heuristics. Elements of domain design will be analysed in detail, including being shown in practical form through the implementation of a domain for the Vehicle Routing Problem with Time Windows.
- The thesis will examine the influence of higher numbers of parameters when performing cross domain optimisation. Two algorithmic heuristic selection methods will be proposed, which use a different number of parameters. Following testing on 4 different problem domains, it will be established whether a higher number of parameters makes it more difficult for a hyper-heuristic to adapt to different problems.
- Evidence will be given of how a hyper-heuristic can use information provided by the problem domain in order to improve results. This will be achieved through the use of adaptive heuristic selection mechanisms that use information about past performance, as well as other factors, to drive selection.
- Additions to the HyFlex framework will be proposed and implemented for the VRPTW domain which will provide a greater amount of data to hyper-heuristics. For all additions, explanations will be given of how they can be used by the hyper-heuristics. In particular, a new hyper-heuristic will be proposed which makes use of the new ‘genotypic’ distance feature that will be added. This hyper-heuristic approach will demonstrate the potential of a distance measure to improve solution quality and demonstrate further the need of a domain to provide the correct tools to a hyper-heuristic.
- A new ‘real-world’ vehicle routing domain will be presented for HyFlex which will show some considerations of domain design for industrial applications,

as well as helping to answer the question of whether a solution can be made more fair whilst maintaining an acceptable level of solution quality. This domain will include several new constraints and features which will allow investigation into the relationship between fairness and solution quality, as well as providing the means to encourage a solution to be fair. Thorough examples of how these features can be used by a hyper-heuristic to promote fairness will be given. It shall also be shown that the set of heuristics used for the VRPTW domain do not perform well on the basic routing problem of the *Travelling Salesman Problem (TSP)*. A new set of operators will be proposed for the domain, along with a discussion of what caused different operators to be successful for different variants of a problem.

1.3 Publications Arising from Work within Thesis

The following publications have arisen during study for this thesis and are related to the work herein. They are presented below in chronological order.

- 1) Edmund K. Burke, Michel Gendreau, Gabriela Ochoa, and James D. Walker. Adaptive iterated local search for cross-domain optimisation. In Proceedings of the 13th annual conference on Genetic and evolutionary computation, pp. 1987-1994. ACM, 2011.

- 2) James D. Walker, Gabriela Ochoa, Michel Gendreau, and Edmund K. Burke. Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. In Learning and Intelligent Optimization, pp. 265-276. Springer Berlin Heidelberg, 2012.

- 3) Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A. Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall et al. Hyflex: A benchmark framework for cross-domain heuristic search. In Evolutionary Computation in Combinatorial Optimization, pp. 136-147. Springer Berlin Heidelberg, 2012.

- 4) Gabriela Ochoa, James Walker, Matthew Hyde, and Tim Curtois. Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework. In Parallel Problem Solving from Nature-PPSN XII, pp. 418-427. Springer Berlin Heidelberg, 2012.

1.4 Thesis Structure

The structure of the thesis is as follows.

- Chapter 2 presents descriptions of the problems to be considered by this thesis, specifically the *Travelling Salesman Problem (TSP)* and variants of the *Vehicle Routing Problem (VRP)*. In addition, the relevant work from the TSP and VRP literature is reviewed and described. Following this, a literature review is performed for the area of *Hyper-heuristics*.

- Chapter 3 describes HyFlex (the Hyper-heuristic Flexible framework), a framework for cross-domain optimisation. Directions are given for the design of both hyper-heuristics and problem domains within this framework. This chapter also introduces a new domain for HyFlex, that of the *Vehicle Routing Problem with Time Windows (VRPTW)*. This domain is described in detail, with the choice of low-level heuristics being explained and 2 new crossover heuristics being introduced.

- Chapter 4 proposes several selection hyper-heuristics for use within the HyFlex framework. Building on previous work within HyFlex, an *Iterative Local Search* algorithmic framework is used and improved. Two different means of heuristic selection are considered, with the aim of investigating what effect larger numbers of parameters can have on cross-domain optimisation.

- Chapter 5 proposes several extensions to the HyFlex framework. For all these extensions, the chapter describes how they can be used to implement new classes of algorithms within HyFlex and improve solution quality. One of these extensions in particular, that of solution distance, is included in a memetic algorithm and tested on the VRPTW. Performance of the algorithm with and without the distance measure is compared and conclusions are drawn about whether it has potential for improving solutions.

- Chapter 6 introduces a new HyFlex domain with the aim of representing constraints and features of routing problems that exist in industrial applications. Specifically, features are introduced which allow investigation into fairness between routes. The chapter describes how these new features can be used to promote fairness. To provide a reliable base for these experiments, a new set of low-level heuristics are described for this domain.

- Chapter 7 summarises the main contributions from each chapter and discusses future directions the work could take.

Chapter 2

Related Work

2.1 Introduction

This chapter will consider the current state of the literature with regards to the overall aims of this thesis. In particular, a formulation of the Vehicle Routing Problem will be provided, along with a review of the vast amount of work that has already been contributed to this problem. By providing this overview of the literature, it can be assured that work in future chapters is making the best use possible of state-of-the-art techniques and that there can be confidence in the quality of the final domain. Furthermore, an exhaustive review of hyper-heuristic research will be provided. Due to the nature of the work, this necessitates a broader consideration of meta-heuristic study and the motivation for hyper-heuristics.

2.2 Vehicle Routing Problem

The widely studied combinatorial optimisation problem, the *Vehicle Routing Problem (VRP)*, was first introduced by Dantzig and Ramser in [46]. The motivation for this problem derives from the scale and needs of the transportation industry, specifically the extent to which small improvements in efficiency could lead to vast savings in expenditure. Indeed, costs attributed to distribution contribute about half of the total logistics costs [118]. Nowhere is the potential for improvement better stated than in [92] where the annual cost for excess travel in the United States alone is estimated to be $45 billion. Many routing and scheduling problems encountered in industry and in personal travel contain a multitude of complex constraints and variables. To name just a few, traffic levels, road-works and even weather conditions can impact on which route should be taken and the overall costs incurred by a journey (be they in time or distance travelled). There are further issues specific to industrial and corporate problems; examples of these are driver shift limits, which may vary between countries, and specified arrival or delivery times for services or goods. Typically, the most widely studied academic VRPs are by comparison quite simplistic, with far fewer constraints. That there still exists such difficulty in obtaining optimal solutions for even these simplified versions of the problem(as will be shown below), goes some way to indicating the difficulty faced in solving such problems. In the following sections, several variants of the VRP will be described with a comprehensive survey of approaches to the problems, with particular focus on the best-performing heuristic approaches. The first of these sections will provide some background in terms of the basics of graph theory and a short description of the *Travelling Salesman Problem*, the relevance of which will be elucidated in later sections.

2.2.1 Graph Theory and the Travelling Salesman Problem

The *Travelling Salesman Problem (TSP)* is one of the oldest and most widely known routing problems. Although its origins are not entirely clear, some studies believe the first formulation of the problem can be found in an 1832 manual for travelling salesmen[112][2] and one of the first academic studies of the problem can be seen in [45]. The basic problem that the TSP represents is that of a travelling salesman who has a specified number of cities to visit and needs to know the optimal order in which to visit these cities in order to minimise the distance to be travelled (and hence reduce time and cost of the journey). The TSP is an NP-hard problem[97]. As the TSP is a special case of the VRP, it shall be described briefly here, with particular focus on certain algorithms that can be successfully applied to the VRP. In order to describe this problem, it is first necessary to outline some fundamental concepts of graph theory.

A *graph*, as a mathematical structure, models the connections between a number of elements. In the context of a graph, these elements are termed *nodes* or *vertices* and the connection between a pair of these *nodes* is referred to as an edge. A complete graph, G, requires both a set of vertices, V and a set of edges, E. Hence, $G = (V,E)$. In an *undirected graph*, an edge is shown as an unordered pair of vertices. In addition, a graph is a *weighted graph* if each edge has an associated cost. A *path* is a certain sequence of a subset of the edges of the graph. A path must satisfy the condition that each edge (with the exception of the first) has as its first vertex the second vertex from the previous edge. For example, $p_1 = \{(a,b),(b,c),(c,d)\}$.

Following on from this, a *cycle* is a special case of a path where a node can be visited more than once. An example of a cycle might be $c_1 = \{(a,b),(b,c),(c,a)\}$.

A *Hamiltonian path* is a path that visits each vertex exactly once and a *Hamiltonian cycle* is a Hamiltonian path that starts and ends at the same vertex.

If considering the Travelling Salesman Problem in graph theory terms, it can be thought of as the problem of finding the minimum cost Hamiltonian cycle for the cities given. The graph for a TSP problem is an undirected weighted graph. Depending on whether an instance of the problem is symmetric or non-symmetric, the edge weights can be calculated in different ways. Typically, if a symmetric instance is being considered, edge weights are calculated as the euclidean distance between two points. That is to say, for cities at the locations i=(x_1,y_1) and j = (x_2,y_2), the following formula is used:

$$dist(i,j) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Where Euclidean distances are used, it can be said that the *triangle inequality* is always satisfied. The triangle inequality states that $dist(i,k) \leq dist(i,j) + dist(j,k)$. That is to say, it is always cheaper (in distance terms) to travel directly to a city, rather than going via one or more other cities.

2.2.2 The Vehicle Routing Problem

The *Vehicle Routing Problem (VRP)* and its many variants can be seen as generalisations of the aforementioned Travelling Salesman Problem. In the VRP, there are a number of *vehicles(routes)* with the task of servicing a set of *customers*. A vehicle will begin at a special case of a customer, named the *depot*, after which it will visit in turn a subset of customers before returning to the same depot to complete the route. In some variants of the VRP, there may be multiple depots to be used. An example of a VRP solution can be seen in figure 2.1.

It can therefore be seen that the TSP is the special case of the VRP where there is only a single route and the depot location can be any city within the route. The objective(s) of the VRP can vary, with multiple possibilities, although the most commonly used objectives are (a)the total number of vehicles utilised and (b)the total distance travelled across all routes. Many of the commonly used measures of quality are described in [89] including measures such as balancing vehicle workload and customer satisfaction. There are many other variants of the VRP, including pick-up and delivery problems, VRP problems that include backhauls amongst others such as the multiple-depot problem previously mentioned.

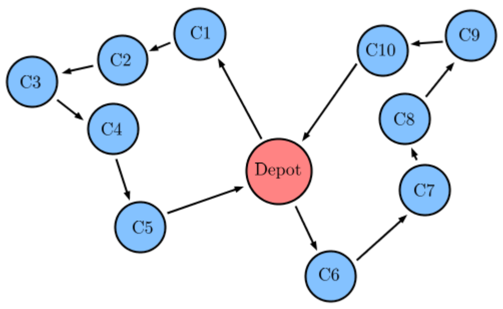


Figure 2.1: An example of a VRP solution.

An extensive description of many of the VRP variants and their formulations is given in [165], building on earlier work in [50]. Two variants in particular have received vast amounts of attention from the academic community, these are the *Capacitated Vehicle Routing Problem (CVRP)* and the *Vehicle Routing Problem with Time Windows(VRPTW)*. It is these two problems that will be the main focus of this literature review.

The Capacitated Vehicle Routing Problem (CVRP) differs from the basic VRP by means of an additional constraint. This constraint adds a maximum *capacity* for each of the vehicles (for the CVRP, this capacity is the same for all vehicles). Each customer in a CVRP instance has an associated *demand* and the sum of all customer demands within a single vehicle/route can not exceed the specified capacity. This capacity constraints mirrors the VRP's industrial applications, where delivery vehicles would be of a certain size and would have a limited amount of space to carry customer goods. The addition of this constraint turns the assignment of customers to vehicles into a *Bin-packing* problem. The bin-packing problem has been extensively studied [100] and large instances of the problem can be solved with minimal use of resources [102].

The Vehicle Routing Problem with Time Windows (VRPTW) adds further complexity to the CVRP. For the purposes of ease of comprehension, it is important to note, for VRPTW instances, the distance value to travel from one customer to another is identical to the time value in the same action. For the VRPTW, each customer has a *start time* and an *end time* representing the times in between which the vehicle must begin servicing them. This is a hard constraint for the TSPTW. However, this does not mean that the vehicle must arrive between these times - it is permissible for the vehicle to arrive before the start time, but to remain inactive for a period (*waiting time*) before beginning servicing a customer.

In addition, each customer now has an associated *service time* which is the time taken to service that customer. Note that servicing does not have to be complete before the end of the time window, but it must have commenced.

As with the TSP, the VRP is an NP-hard problem [97] although it would be fair to say that the VRP is a considerably harder problem to solve. As such, heuristic approaches to solving the problem have been some of the most successful. Traditionally, heuristic approaches are better able to handle larger instances and are more flexible where complex constraints are concerned. The following sections will examine in detail the most effective approaches, both for the CVRP and VRPTW.

Constructive Heuristics

A *constructive heuristic* is a heuristic which creates or builds an initial solution. This is usually built one step (i.e. one city/customer) at a time. While constructive heuristics can generate solutions of a reasonable standard, they are more commonly used to generate solutions that can then be iteratively improved by other classes of heuristic.

One of the first, and most well-known, constructive heuristics is the *Clarke and Wright Savings Heuristic.* This heuristic was first proposed in 1964 [36] for the basic VRP. The first stage of the savings heuristic is to assign a separate route for each customer in the instance. Then, iteratively, routes are selected to be combined; chosen by their potential for greatest savings in cost. The saving to be gained from connecting two customers i and j which reside at the end of two different routes is calculated by the following formula; $s_{ij} = dist_{i0} + dist_{0j} - dist_{ij}$ where 0 is the depot city and *dist* is the euclidean distance calculation mentioned previously. Although it is the new edge resulting in the greatest saving that is chosen, this new solution must still be feasible. In [171] and [62], an improvement is suggested whereby a new formula is used to calculate savings - $s_{ij} = dist_{i0} + dist_{0j} - \lambda dist_{ij}$. Here, the new parameter λ is used to affect the 'shape' of a route and reduce the significance of distance from the depot. Further improvements are considered in [114] where 6 implementation possibilities for the savings heuristic are considered. Paessens [125] also proposes an improvement which results in a reduction in storage requirements and computation time. Parallel methods of implementing the heuristic have also been proposed, such as the one in [73]. Despite the popularity of the method, results for the savings heuristic are generally poor, as can be seen in the survey by Cordeau et al[38]. The advantage to the method is the speed at which it generates solutions.

The 1987 paper from Solomon [151] suggest an extension to the savings method that can be used for the VRPTW. The proposed approach uses as its base the parallel implementation of Golden et al [73]. To adapt the method for the VRPTW, Solomon proposes a parameter, W, to be set as the maximum acceptable waiting time for edges that are being considered. From testing on a dataset introduced by the same paper, it is shown that the savings heuristic is very poor in comparison to other constructive heuristics for the VRPTW.

The Solomon paper[151] also described a number of other solution construction heuristics which operate by iteratively inserting nodes into a solution. The first of these is a *Time-oriented Nearest Neighbour* heuristic. A nearest neighbour heuristic operates by, at each stage, selecting a node (city) for insertion that is closest (as determined by some distance measure) to the location it is to be inserted. For this VRPTW method, closeness is measured in relation to the previous city to be inserted with the cost calculated as a measure of both the proximity in terms of distance and in terms of length of time between a potential servicing - as dictated by each city's time window constraint. Solomon also proposes 3 insertion heuristics, similar in concept to a method proposed in [59], which operate in a similar way to one-another. The basic idea is that a route will be initialised with the customer evaluated as furthest from the depot. Then, iteratively, all possible insertion positions for each remaining customer are considered. If there are no feasible insertions, a new route is initialised and the process repeats. The difference between the 3 heuristics is the means by which they judge the worth of an insertion position. The most successful of the 3 is the *I1* heuristic which uses a weighted sum of the distance and time it will take to visit a city when selecting insertions.

There have been a number of improvements proposed to Solomon's *I1*[151] heuristic. In [56], Dullaert and Bräysy propose a modification to the *I1* heuristic that calculates a higher time value for potential insertions. Using this modification, significant improvements are obtained over Solomon's *I1*, although, these improvements are mostly seen on instances with a lower customer-to-route ratio. A parallel implementation of Solomon's algorithm is proposed in [129]. Comparisons to the *I1* heuristic show that the parallel approach yields minor improvements in how many routes are used for instances where customers are 'randomly' located. However, the inverse is true for 'clustered' instances. Ioannou et al. [87] propose new criteria for the selection and insertion of cities and apply their method to a number of academic problems, as well as an industrial application.

Another approach taken to constructing solutions for the VRP has been to split the problem into two elements, *clustering* and *routing*. These form a class of algorithms known as *Cluster first, route second* algorithms and also the less common approach of *Route first, cluster second* algorithms. One of the most well-known approaches of this type is the *Sweep algorithm*. From a concept first discussed in [170], Gillet and Miller [68] propose a method named the *sweep* algorithm. The sweep algorithm uses polar-coordinate angles to assign customers to routes and produces results that are superior to the savings method, but at a higher computational cost. It should be noted that, in [68], an iterative improvement method is applied to the solution following creation of the initial solution. The analogy of petals of a flower is often used when describing methods of the cluster-first, route-second class. In this analogy, each route is a petal on the plant. The term was first used in a VRP context by Ryan et al. in [142] which introduces an algorithm operating in a similar way to the sweep method, but differs by also considering non-petal routes. Another method described in terms of the petal analogy is the algorithm proposed by Renaud et al. [137] which combines elements of the sweep method and column generation methods to acheive strong results in reasonable computation times.

The concept of the sweep algorithm is one extended by Solomon [151] for the variant of the VRP with time windows. The same basic method is used initially as in the sweep method [68] to partition customers into routes. However, time window constraints mean that some customers can not feasibly be inserted into the route. The algorithm repeats the stages of sweep and route until all customers have been inserted. Experiments performed in the Solomon paper [151] show that this modified sweep algorithm outperforms the savings method but returns worse results than the *I1* insertion algorithm.

An alternative to the cluster-first, route-second class of algorithms is the *Route-first, Cluster-second* class. One of the first examples of this algorithm for the VRP is the work by Beasley [6]. In his article, Beasley first forms a 'giant tour' of all cities in a single route. This route can then be treated as a travelling salesman problem and improved with the relevant methods. Following this, the route can then be split into the appropriate number of individual routes. Beasley uses Djikstra's algorithm for partitioning the giant route and states that it is computationally very quick to solve. Results given in the paper show a slight improvement on the savings method. However, given the presence of an iterative improvement method in the Beasley algorithm, the comparison must be made with a pinch of salt. In [12] and [11], results for methods of this class are discussed and it is concluded that performance for these methods is similar to performance for the bin-packing heuristic *Next-fit*. These methods are not as effective for instances of the VRPTW as the time-window constraint means that the initial giant route can not be treated as a travelling salesman problem.

Solution Improvement Heuristics

Often heuristic and meta-heuristic techniques for solving vehicle routing problems will employ two main stages. Firstly, one of the methods described in the previous section will be used to create an initial solution that can be used as a seed solution. Some methods will create multiple initial solutions, genetic algorithms are one example. Following this, some form of improvement heuristic/s may then be applied, often iteratively, until some form of stopping criteria has been met. These improvement heuristics will typically involve making a simple modification or move to the solution in order to form a new solution. This section will consider these solution improvement methods for various forms of vehicle routing problems.

One class of these heuristics concerns the improvement of distance (or time) within individual routes. Where time window constraints are not present, a single route can be considered as a travelling salesman problem - a problem for which many efficient improvement methods exist. The most well-known (and indeed most successful) of these methods is the set of *k-opt* or *k-exchange* algorithms. A k-exchange move is a move where k edges in a route are replaced with k new edges. A route is *k-optimal* if there is no possible k-exchange move to be made that can improve the route. This approach was first proposed by Lin in [98] where particular focus is put on *2-opt* and *3-opt* algorithms, although the first description of a *2-opt* algorithm can be found in [42]. Good results are obtained by Lin on small instances of the TSP. Usage of larger values of k such as 4 and 5 is present in work by Christofides and Eilon [35] where strong results are obtained on larger instances of the TSP (up to 500 cities).

A problem with the above methods can often be how to choose what value of k should be used in order to find the correct compromise between attaining acceptable solution quality, in a reasonable amount of computational time. Lin provides a solution to this problem in [99] where a generalised version of the above algorithms is proposed which adaptively selects values of k to be used at different stages of the process. This method manages to obtain optimal results for travelling salesman problems up to a size of 110 cities at reasonable time cost. Several improvements to the implementation of Lin's method are proposed by Helsgaun in [76] which allow savings in time and great savings in solution cost. The new implementation obtains optimal solutions on TSPs up to the size of 13509 cities. Another operator of this type is the *Or-opt* method proposed by Or in [120]. The Or-opt algorithm relocates sequences of consecutive cities and is applicable to both the CVRP and the VRPTW.

All of the above techniques are concerned with improving individual routes, whether this improvement is in terms of distance travelled or time taken. Many of these techniques have been derived from methods for the travelling salesman problem. Another way that solutions can be improved for vehicle routing problems is by making modifications that involve several routes. One successful heuristic to fall into this category is the *2-opt** method of Potvin and Rousseau [130]. Despite sharing a similar name to the previously described 2-opt heuristic, there are few similarities between the methods. The 2-opt* approach consists of swapping the end sections of two routes. As well as the potential to reduce distance/time, this method can also reduce the number of routes in the special case where the end section of one of the routes is merely the depot and the end section of the other route is the entire route after leaving the depot. The Potvin and Rousseau paper [130] also presents a hybrid approach combining 2-opt* and or-Opt which produces competitive results for the VRPTW.

In [143], Savelsbergh proposes a number of heuristics which move customers between routes. These are:

- **Relocate** This heuristic moves a single customer from one route to another.
- **Exchange** For this method, a single customer is selected from each of two routes. These customers are then swapped into each other's routes. However, they aren't necessarily inserted into the same position within the route as the previous customer.
- **Cross** The cross heuristic attempts to modify edges in a solution in such a way as to remove instances of *crossed edges* within a solution. Although time window constraints can render the triangle inequality meaningless for instances of the VRPTW (that is to say, the constraints can often mean it is impossible to find a feasible solution without any crossed edges), it is still the case that crossing edges should be avoided where possible.

Building from this work, Van Breedam [166] proposes a classification system for move operators for vehicle routing problems. In this classification, the 3 above moves are included (*relocate*, *exchange* and *cross*) as well as a fourth category called *mix* which allows for a combination between exchange and relocate. In [132], Prosser and Shaw utilise the 3 heuristics from the Savelsbergh [143] paper as well as single-route methods such as 2-opt in their approach. Despite using a simplistic heuristic framework, testing on the Solomon [151] benchmark VRPTW instances produced strong results, including 4 new best-knowns. Analysis of the effectiveness of individual heuristics by the Prosser and Shaw paper [132] indicated that the relocate heuristic was the strongest.

An extension to the relocate operator is proposed by Gendreau et al. in [65]. The heuristic, named *GENI*, also relocates a customer from one route to another. The method differs in the way that the customer is inserted into the new route. For the relocate operator, the customer is inserted in between consecutive customers on that route. GENI removes the requirement for these customers to be consecutive by performing local re-optimisation. In [154], Taillard et al. propose a new solution improvement heuristic called *CROSS* (not to be confused with the Cross method of [143]). This heuristic operates by switching sections of separate routes, whilst maintaining the sequence of cities. Due to the nature of time window constraints, the action of maintaining the sequence of cities is one that is particularly beneficial for instance of the VRPTW.

In [70], Glover proposes the application of an *Ejection Chains* method for travelling salesman problems. Ejection chains operate by performing a sequence of compound moves, where one move can cause another move until some stopping condition is met. This method can be applied to the VRP by repeatedly removing a customer from one route and attempting insertion into another route. If insertion is not possible, due possibly to capacity or time window constraints, another customer is removed from the target route in order to create space for the customer to be inserted. This process is repeated until a customer can be inserted without the need to eject another customer. Thompson and Psaraftis extend this concept in [164] with their *Cyclic Transfer* algorithm. This method is similar to the ejection chain method, but attempts to shift several customers from each route at one time. Using this approach, strong results are obtained on benchmark VRP instances.

A different approach to improving VRP solutions is proposed by Schrimpf et al. in [144], called the *Ruin-recreate* method. The basic idea behind ruin-recreate has two stages:

- **Ruin** Several customers are removed from the solution. Using a method named *Radial ruin*, customers are chosen to be removed according to their proximity in either distance or arrival time to an arbitrarily chosen base customer.

- **Recreate** Following removal of customers in the ruin stage of the algorithm, these customers are then to be re-inserted into the solution. Iteratively, a customer is chosen at random from the set of unrouted customers. The 'best insertion' position for this customer is then calculated and the customer is inserted into this position. A position is determined to be a 'best insertion' position through its objective function value in comparison to other possible insertions. The insertion is only performed if the resulting solution is feasible.

The application in [144] of the ruin-recreate method to the VRPTW produces results which are stronger than previous heuristic approaches.

Meta-heuristic Approaches

The term *Meta-heuristic* is used to describe a wide range of search methodologies. They will often describe the entire search process, including which construction heuristic is used, the choice and use of improvement heuristics and solution acceptance criteria, which determines whether a solution is 'kept' following application of a heuristic. For many combinatorial optimisation problems, meta-heuristics can be very powerful and provide a flexibility absent from many exact methods. Thus, they can be particularly effective for variants of the VRP with time windows where the search space is more constrained. Many meta-heuristics can also be easily understood on a conceptual level. Indeed many are analogous to methods found in nature or other scientific disciplines. It is these methods which shall be considered first.

Ant Algorithms The concept of an *Ant Algorithm*, or *Ant Colony Optimsation(ACO)* method as it's also known, was first proposed by Dorigo et al. in [53]. The later paper by Dorigo et al. in [52] presents a detailed study of the ACO method and how it can be applied to many well-known combinatorial optimisation problems. The algorithm is inspired by the real-life behaviours of colonies of ants with particular focus on the way in which ant colonies determine the shortest path to a location, e.g. a source of food. This is achieved through the use of pheromones that are left on trails travelled by ants. Higher levels of pheromones will encourage more ants to take those trails. Shorter trails accrue greater levels of pheromones as they take less time to traverse and hence the amount of time for an ant to complete a journey both ways is shorter and pheromones are also built up more quickly.

The first instance of an ant algorithm being applied to the VRP can be seen in [17] where Bullnheimer et al. apply the method to the capacitated vehicle routing problem. Their method generates solutions by sequentially selecting cities using their relative pheromone strength, before improving these solutions through the use of the 2-opt algorithm. Reasonable results are obtained when compared to other meta-heuristic methods. This method is improved by Bullnheimer et al. in [18] where reduced candidate lists are used for the selection of cities. Slight improvements are found in both run time and solution quality.

Gambardella et al. [60] use an ACO method with multiple colonies to solve the vehicle routing problem with time windows. Competitive results are achieved on benchmark VRPTW instances with new best-known solutions on some instances. The work of Bell and McMullen in [7] also uses multiple ant colonies to solve vehicle routing problems. Further, it considers different sized candidate lists, as with the work in [18]. Using this approach, strong results are obtained on smaller instances of VRPs. However, performance is poorer for larger problems. In [134], Reimann et al. extend their previous work of [135] with a method called *D-Ants*. A modification of the aforementioned Savings algorithm is used to generate solutions, before a local search stage comprising 2-opt moves and customer exchanges is performed. The problem is also simplified by being decomposed into a number of smaller sub-problems which can be solved more easily. This approach produces strong results on a number of VRPs with varying constraints, including quite large instances. In [51], Doerner et al. propose a parallel version of the D-Ant algorithm for the VRP. Through this parallelisation, an improvement in speed is obtained.

An approach by Hu et al. in [83] proposes a method for adjusting pheromone levels. When used in combination with solution improvement methods, reasonable results are found in short time periods for the VRPTW. A hybrid method is presented in [172] where ant colony optimisation is used in conjunction with a dynamic sweep method to generate solutions for instances of the VRPTW in the Solomon dataset. Another example of an ACO method being used to solve the VRPTW can be found in [159].

Genetic Algorithms A *Genetic Algorithm (GA)* is a population-based approach that is designed to partially reflect that natural process of evolution. Key stages in these algorithms are often combination of solutions and mutation of solutions. The idea of reproducing this natural method in an artificial system was first discussed in [78]. The majority of GAs for vehicle routing problems have been designed for the VRPTW subset of VRPs. However, there are some examples for the CVRP. One such example can be found in the work of Baker and Ayechew [4] where two GA variants are proposed. The first is a basic GA, newly tested in the context of the CVRP, and the second is a hybrid of this first GA with other neighbourhood search methods. Results obtained using this second method are comparable to those obtained with other leading meta-heuristic algorithms.

Berger and Barkaoui also present a hybrid GA for the CVRP in [10]. This approach maintains two populations which are evolved in parallel. Further, a number of local search methods from algorithms for the VRPTW are utilised. Testing on benchmark problems shows strong results for the CVRP using this method. The work of Prins in [131] also uses a hybrid approach which utilises local search optimisation methods to augment the basic GA. By using this method, the algorithm forgoes the need for a repair mechanism that is present in many other GAs for the VRP. This new approach proves to be very effective on CVRPs, particularly on larger instances of the problem. Kubiak [94] focuses on recombination operators for VRPs. These operators attempt to preserve distance within solutions. Strong results are obtained when these operators are used in the context of a *Genetic Local Search.*

In [1], Alba and Dorronsoro propose the application of a variant of the basic GA to the capacitated vehicle routing problem. The variant, named the *Cellular Genetic Algorithm* includes a population where population members can only interact with a subset of the rest of the population, their 'neighbours'. By dividing the population in such a way, a balance between intensification and diversification is achieved whereby each neighbourhood can be optimised to a locally optimal level. When tested on a large set of instances including most benchmark problems within the CVRP literature, the best-known solutions are either matched or improved in 80% of cases. In the work of Mester et al. [107] the focus is on the mutation stage of the algorithm. Here, a multi-parametric mutation method is used to achieve best-known or better results in 42% of tested instances. Another approach can be found by Mester and Bräysy in [106]. Again, a hybrid approach proves successful, where a *Guided Local Search* is used with a genetic algorithm in a two stage approach. Results from this method match or better best-known solutions in 70 of 76 tested instances.

The first example of a GA for the vehicle routing problem with time windows (VRPTW) can be seen in the work of Thangiah et al. in [161] with an extended description in [160]. This genetic algorithm operates in the context of a cluster-first, route-second method where the GA is used to generate desirable clusters. Insertion heuristics are used to actually generate the routes before solution improvement methods are applied. The application of two evolutionary approaches, those of Rechenberg [133] and Schwefel [147], is proposed for the VRPTW by Homberger and Gehring in [80]. Three elements are used in [80] to define the two algorithms; these are the initialisation, the selection of values for two parameters, (μ,λ), and the termination criteria. For both algorithms, the construction heuristic is a stochastic method based upon the savings heuristic of Clarke and Wright [36]. The termination criteria is also the same for both and is in the form of a time limit for the search. The main difference in algorithms is in the choice of parameter values. The parameter μ is initially the size of the starting population. At each generation, a number of offspring, λ are generated, under the condition that $\lambda > \mu$. Values of (8,50) and (45,450) for the two algorithms respectively show the difference in population sizes for the two methods.

Gehring and Homberger present another evolutionary method for the VRPTW in [63]. It is a two-stage approach, where initially an evolutionary method is used to reduce the number of routes in a solution. The second stage utilises a *tabu search* method (see later) to improve the total distance travelled. Both a sequential and parallel version of this method are considered. Strong results are achieved on the Solomon [151] set of benchmark instances. Gehring and Homberger extend their work in [64] by developing the parallelisation of the previously described method. The parallel method operates by concurrently applying differently parametised meta-heuristics to the search space and then exchanging solutions to combine information. Results imply that this could be a useful method for large scale VRPs. A parallel approach is also considered by Le Bouthillier and Crainic in [15]. In that approach, a *solution warehouse strategy* is used where multiple threads can exchange information about the best solutions.

Tan et al. [158] achieve strong results on the Solomon benchmark [151] with the use of a *Messy Genetic Algorithm*. The concept of a messy GA was first proposed in [71] by Goldberg et al. and is a form of a GA which uses strings of a variable length as opposed to the standard fixed length. The algorithm in [158] uses a random technique to generate an initial population and a 1-point crossover.

In what is another hybrid method, Ho et al. [77] combine a GA with a *Tabu Search*. By combining desirable qualities of both methods, the hybrid approach outperforms the individual techniques. The [77] method also utilises several local search methods (2-opt, 2-opt*, exchange, relocate) for solution improvement. A further hybrid method is presented by Jung and Moon in [90], where solution improvement methods are used to augment a genetic algorithm. Specifically, 3 solution improvement techniques (or-opt, cross, relocate) are used. The insertion heuristic, *I1*, of Solomon [151] is also used to create initial solutions.

Tabu Search The concept of a *Tabu Search* algorithm was first introduced by Glover in [69]. A tabu search is a meta-heuristic which repeatedly makes neighbourhood moves to attempt to improve a solution. Its novel feature is the inclusion of a *tabu list* which maintains a list of either moves or solutions states which are temporarily forbidden within the search. One example of how this can be used is when a neighbourhood move improves a solution. The previous solution can be added to the tabu list to ensure that the search doesn't cycle. A tabu list can also be maintained of operators that are performing poorly at that point in the search. For elements in a tabu list, whether solution state or operator, they will remain in the list for a certain amount of time, termed the *tabu tenure*.

The first application of a tabu search to the capacitated vehicle routing problem can be seen in the work of Osman [121], where 2 approaches are applied to the CVRP. The first approach is a tabu search implemented with a new data structure which the author reports reduces computation time by up to 50%. The second approach combines this tabu search with a *Simulated Annealing* method for solution acceptance. Simulated Annealing is a method which determines whether or not a solution is accepted following a neighbourhood move. All improving solutions are accepted and some deteriorating solutions are accepted with a low probability. Using these approaches, Osman [121] achieved (at the time) new best-known results on instances from the literature.

Taillard [153] proposes an approach for the CVRP that attempts to diversify the nature of moves applied. Taillard observes that the traditional approach of forbidding reverse moves only can lead to many moves being performed close to the depot. To remedy this, in [153], the tabu list also includes moves which have been frequently made, applying an approach that had previously been used in a scheduling problem in [155]. This tabu search approach is used in the context of an algorithm which decomposes the VRP into smaller sub-problems and performs strongly on a new set of proposed instances.

Also for the CVRP, the tabu search method of Gendreau et al. [66] which penalises vertices which are frequently moved via a penalty term in the objective function. Another point of interest regarding the tabu search of [66] is that moves are permitted to enter the infeasible search space with respect to capacity and distance constraints. Further, the algorithm differs in the implementation of a tabu search in that a tabu list is not explicitly maintained. Instead, each move is assigned a tag indicating its current tabu tenure. Testing on benchmark instances shows a strong performance using this method and several new best-known solutions.

The first application of a tabu search to the vehicle routing problem with time windows was by Garcia et al. in [61]. A parallel algorithm is used, where multiple moves are applied at the same time to the incumbent solution. The moves considered are two-opt* and or-opt moves and the tabu search element of the algorithm is that following a move, the inverse of that move is added to the tabu list. Using this approach on Solomon's benchmark instances [151], improved results are achieved over Solomon's *I1* heuristic [151].

An approach by Rochat and Taillard [139] uses a post-optimisation technique to improve an initial tabu search algorithm. Using this approach, new best-known results are gained for VRPTW. In Taillard et al. [154], a tabu search method is applied to the vehicle routing problem with soft time windows. In this problem, penalty values are applied to the objective function for cases where the time windows are violated. The neighbourhood move used in [154] is the CROSS heuristic proposed in the same paper. The tabu search method operates on decomposed problems which include subsets of routes. Potential moves are added to the tabu list if they do not yield an improvement in the objective function value. Badeau et al. [3] provide an extension to the work of [154] by using the same method, but in a parallel algorithmic framework. This approach leads to results of a comparable quality to [154] but reduces the amount of computation time needed.

A *Reactive Tabu Search* is proposed by Chiang and Russel in [34]. The tabu search is reactive as it adapts the size of the tabu list so as to ensure that cycles are avoided whilst not constraining the search to a great degree. The method is tested on real-world routing problems as well as standard benchmark VRPTW instances.

Another parallel tabu search method is proposed by Schulze and Fahle in [145]. As with [66], the method of Schulze and Fahle [145] also allows solutions to temporarily enter the infeasible search space. A customer shift operator is used as a neighbourhood move operator. This approach is tested on the Solomon [151] benchmark and returns some of the strongest results using a tabu search method.

Tan et al. [157] introduce a tabu search algorithm for the VRPTW. The method includes a tabu list which maintains two types of elements. The first is recent neighbourhood moves that have been made. The second is recent solution states. The tabu search yield strong results when compared to a genetic algorithm proposed in the same paper. In [39], Cordeau et al. also present a tabu search for the VRPTW. In addition, the method is tested on the periodic VRP and the multi-depot VRP. A sweep algorithm [68] is used to initialise solutions. This tabu search also allows solutions to be in the infeasible search space. The relocate and GENI heuristics are used as solution improvement methods. Results from using this approach generate new best known solutions for the tabu search algorithm on the VRPTW.

2.3 Hyper-Heuristics

Heuristic and meta-heuristic methods have been providing high quality solutions for a variety of problems for many years. Their successes have been found in diverse and well-studied academic problems as well as in complex and constrained real-world problems. However, whilst often effective for solving the problems they are designed for, meta-heuristic approaches are often not able to adapt to changes in a problem structure or even to different problem instances with the same structure. Hard-coded parameter values and specific algorithmic structures can mean that methods are tuned, intentionally or otherwise, to work well on specific problem structures and instances. In industrial applications, an ability to adapt to a changing problem space with little expense can be very important. The examples are numerous. Consider a delivery truck which has to change its route due to a closed road. Or a breakdown in machinery requiring jobs to be re-assigned to other machines. In these situations, a lengthy tuning process to generate a new solution would be highly undesirable.

This is an issue which *hyper-heuristics* attempt to address. Hyper-heuristics operate on the space of heuristics, rather than the space of solutions. Practically, this means that a hyper-heuristic doesn't have specific knowledge of the problem being operated on; instead it manipulates a set of heuristics in such a way as to improve a solution. In this way, hyper-heuristics are a more general approach to solving hard combinatorial optimisation problems. Before reviewing the work in this area, it is important to consider a definition of hyper-heuristics.

The term ‘hyper-heuristic’ was first used in [49] to describe the combination of a number of AI methods. However, its first use in the context described above can be seen in the work of Cowling et al. [41] where it’s used to describe an approach to solve a personnel scheduling problem. In that paper, a hyper-heuristic is described as ‘a heuristic to choose heuristics’. To find the first full definition/classification of hyper-heuristics it is necessary to skip forward to the paper of Burke et al. in [28]. The classification in [28] contains 2 parts. The first describes how the hyper-heuristic manipulates the heuristic search space and classifies hyper-heuristics as one of the following. One option is a **Heuristic Selection** hyper-heuristic which selects heuristics to be used in a search. The other option, **Heuristic Generation**, represents hyper-heuristics which generate new heuristics by combining elements of previous heuristics. The classification for the options just described is extended further in [28] by stating whether the hyper-heuristic operates on construction heuristics or perturbation heuristics.

The second part of the above classification describes how a hyper-heuristic uses feedback from the search. A hyper-heuristic can be classified into one of the 3 categories below.

- **Online Learning** This is a hyper-heuristic which uses information gained during the search to adapt the selection/generation of heuristics. For example, by observing that a certain heuristic repeatedly improves the solution, a hyper-heuristic may choose to increase the probability of that heuristic being selected.
- **Offline Learning** This type of hyper-heuristic uses results from a set of instances to determine a set of rules for heuristic selection or generation. This is performed before the start of a search.
- **No feedback** This is a hyper-heuristic which doesn’t use any form of feedback to select or generate heuristics. An example from this category of hyper-heuristic would be a method which selected heuristics at random or applied heuristics in a predefined sequence that wasn’t based on any feedback. Although no feedback is used, it can still be considered a hyper-heuristic as it fits the previously given definition of a heuristic which selects other heuristics - even if this selection is performed randomly.

The hyper-heuristics to be considered within this literature review are in the category of **Heuristic Selection** hyper-heuristics. All 3 options for feedback (or lack thereof) to hyper-heuristics will be considered.

As was mentioned above, the first work to propose a selection hyper-heuristic using the term 'hyper-heuristic' was the work of Cowling et al. [41]. The work considers 3 types of hyper-heuristic methods for solving a real-world personnel scheduling problem. These 3 types of hyper-heuristic in [41] are:

- **Random Heuristic Selection** There are 3 hyper-heuristics within this set which all select low-level heuristics in a random manner. The 3 hyper-heuristics are:
  - *SIMPLERANDOM* This hyper-heuristic repeatedly applies a random low-level heuristic.
  - *RANDOMDESCENT* This hyper-heuristic selects a random low-level heuristic and then repeatedly applies this heuristic until no improvement in solution quality is found.
  - *RANDOMPERMDESCENT* This hyper-heuristic first defines a random ordering of the set of low-level heuristics. Then, each heuristic is repeatedly applied in turn until it does not yield an improvement in solution quality, at which point the next heuristic is applied.
- **Greedy Heuristic Selection** This approach is similar to that name *Best Improvement*. At each iteration, all low-level heuristics are independently applied to a solution, with the heuristic yielding the best result being selected and applied.
- **Choice Function Heuristic Selection** The choice function hyper-heuristic selects a heuristic at each iteration by using online learning to assess the benefits of each heuristic. Three measures of heuristic performance (*f1*, *f2* and *f3*) are used in the choice function. These are:
  - *f1* The first measure, *f1*, considers the improvements in objective function achieved from applications of the heuristic under consideration.
  - *f2* The second measure, *f2*, considers how the heuristic has previously performed when applied immediately following the previous heuristic to be applied. In other words, this measure considers the performance of pairs of heuristics.
  - *f3* The third measure, *f3*, considers the time taken to apply the heuristic under consideration.

Testing of these hyper-heuristics on the personnel scheduling problem resulted in all hyper-heuristic approaches beating the previous greedy method that had been used to solve the problem. Of the hyper-heuristics, the choice function provided the strongest results, implying that there is value in using information gained from previous heuristic applications to select future heuristic applications.

In the short time that hyper-heuristics have been studied, there have already been many hyper-heuristic approaches to different combinatorial optimisation problems. Further, there have been approaches that apply the same hyper-heuristic to several problem domains without manual tuning in between runs. Recent hyper-heuristic work will now be considered, according to problem domain.

2.3.1 Personnel Scheduling

A description has been given above of the first hyper-heuristic approach [41] for a personnel scheduling problem. Han, Kendall and Cowling [75] propose a hyper-heuristic in a genetic algorithm framework. The proposed method uses variable length chromosomes which represent sets of low-level heuristics. Performance measures such as heuristic performance and the compatibility of a heuristic with other low-level heuristics are used to determine whether or not heuristics are removed from chromosomes. The approach is tested on a trainer scheduling problem and provides stronger results than a previous method which had fixed chromosome lengths. An extension to this method is proposed by Han and Kendall in [74]. Here, the addition or removal of heuristics from the adaptive length chromosomes is better guided as the algorithm identifies when individual chromosomes are of a length that can be considered too short or too long. This addition provides an improvement in solution quality when tested against the previous version.

In [40], many hyper-heuristic approaches are considered and applied to a real-world personnel scheduling problem. In this work, a number of variations of a hyper-heuristic called *Peckish* are proposed. Through this approach, a balance is sought between intensification and diversification and is achieved by selecting low-level heuristics in both an intelligent manner (using data about previous success) and a random way. Another set of hyper-heuristics based on the tabu search meta-heuristic are also described in the same paper. For these methods, a tabu list is maintained. If a heuristic improves a solution, the resulting new solution is always accepted. If the solution does not improve, it is only accepted if the heuristic is not in the tabu list. Finally, a group of simple random and greedy hyper-heuristics are also considered. Experiments on the aforementioned trainer-scheduling problem show strongest results for the tabu search methods, particularly when they are starting from a worse initial solution.

Another tabu search method is proposed for rostering problems by Burke et al. in [29]. For this tabu search, each heuristic has a rank associated with it. When a heuristic is applied and it yields an improvement in objective function value, its rank is increased. Conversely, it is decreased if an application worsens the solution. There is also a tabu list maintained. If an application of a heuristic worsens a solution, the tabu list is emptied and heuristic is inserted into the list. The tabu search hyper-heuristic is compared to a genetic algorithm and proves more able to produce feasible solutions, although at a worse cost.

In [109], Misir et al. propose a hyper-heuristic for the home care scheduling problem. The proposed method keeps a dynamic heuristic set, by means of a tabu search mechanism. Low-level heuristics are each given a quality index, *QI* which measures their performance over the search. Performance is measured by 3 factors: i) the number of new solutions found by the heuristic, ii) the fitness improvement achieved relative to execution time, iii) the fitness deterioration relative to execution time. 3 solution acceptance methods are also used in variants of the hyper-heuristic. Testing shows that the hyper-heuristic is able to accurately detect the performance differences of the heuristics.

2.3.2 Timetabling Problems

In [21], Burke et al. present an ant algorithm hyper-heuristic for the project presentation scheduling problem. In this method, ants visit sequences of heuristics, laying appropriate amounts of pheromones depending on whether the solution has been improved or not. The ants operate on 8 low-level heuristics for the problem and testing uses between 3 and 5 ants. This approach is compared to the *Simple Random* and *Choice Function* hyper-heuristics and produces generally superior results.

An ant algorithm hyper-heuristic is also presented for the travelling tournament problem in [33]. In [33], Chen et al. use the same basic concept of ants 'visiting' heuristic and laying pheromone trails based on performance as is proposed in [21]. However, due to the differing nature of the problem, in this case the ants may cycle back to previously visited points. In addition, a measure called *visibility* is used to inform ants of how far they are from certain locations. This visibility is used in this method to represent the execution times of heuristics. The method performs as well as the best-known methods on smaller instances; however the results are further away for larger instances.

Comprehensive experiments are carried out in [13] by Bilgin et al. on exam timetabling problems. Multiple hyper-heuristic combinations are tested with 7 heuristic selection mechanisms and 5 solution acceptance methods. The results over a range of benchmark objective function indicate that while some hyper-heuristics will perform better on certain functions, none of the tested approaches can beat all the other methods on all functions. Never-the-less, when considering the results as a whole, the most successful approach seemed to be the choice function with a monte carlo solution acceptance mechanism.

In [122], Özcan et al. propose a *Late Acceptance* hyper-heuristic for an examination timetabling problem. Late acceptance [23] is a solution acceptance methodology. Rather than comparing the objective function value for a new solution to that of the current solution, instead the new value is compared to a value from L moves previously. Testing using this method showed that it performed well when used with a *simple random* hyper-heuristic. However, performance was poorer when late acceptance was used with hyper-heuristics which had an element of re-inforcement learning within their selection mechanism.

Another method of solution acceptance is used in the work of Özcan et al. in [124] where the *Great Deluge* solution acceptance mechanism is used with a re-inforcement learning hyper-heuristic on an examination timetabling problem. The great deluge method [55] simulates a rising water level which represents the acceptance rate for solutions. Using this method, it is far easier for worsening solutions to be accepted near the beginning of the search and towards the end hardly any deteriorating solutions are accepted. It is found that the great deluge method works better with the re-inforcement learning technique than with a simple random method.

In [48], Demeester et al. consider a number of examination timetabling instances, including a new real world instance. The performance of tournament-based hyper-heuristics is analysed and it is found that significantly better results are achieved using a hyper-heuristic than were achieved manually for the new instance.

2.3.3 Vehicle Routing Problem

In [126], Pisinger and Ropke introduce a solver which is able to operate on many variants of the vehicle routing problem (for example, multiple depot VRP, capacitated VRP, VRPTW). The algorithm used to solve these problems is an *Adaptive Large Neighbourhood Search (ALNS)* and is based on a the *Large Neighbourhood Search(LNS)* of Shaw [150]. The ALNS method adds an adaptive layer to the LNS which selects heuristics which can either intensify or diversify the search. The method performs well on all the tested VRPs and shows an ability to adapt to varying problems.

Meignan et al. [105] present a co-operative hyper-heuristic for the vehicle routing problem. A set of agents attempt to improve a solution in parallel. Heuristics are selected through a re-inforcement learning technique. This technique takes into account heuristic performance as observed by that agent. However, the agents also communicate, meaning that information gathered from other agents can be used to select heuristics. Strong results are obtained on a number of VRP instances.

In [108], Misir et al. provide a hyper-heuristic approach to some real-world instances of a ready-mixed concrete delivery problem. The approach used is a simple random heuristic selection mechanism with a new solution acceptance method, named *adaptive iteration limited list-based threshold accepting with a fixed limit (AILLA-F).* The AILLA-F acceptance method operates by maintaining a list of best solutions in a window of size l. This method is compared to 4 other solution acceptance methods when combined with the same heuristic selection mechanism. The AILLA-F method was shown to provide the strongest results.

2.3.4 Cross-domain Optimisation

As was discussed above, the development of hyper-heuristics aims to raise the level of generality for problem-solving. This means that algorithms operate on problems without knowing any problem-specific information and operate solely on the heuristic search space.

The *Hyper-heuristic Flexible(HyFlex)* framework allows exactly this type of algorithm to be implemented. The framework, first proposed by Burke et al. in [20] and with a full description in [116], provides a number of problem domains which hyper-heuristics can be tested on by means of a common interface. The hyper-heuristic can access information about how many heuristics a problem domain has and what category they fall under (mutation, local search, ruin-recreate, crossover) and can also access objective function values for the population of solutions. However, they do not have to know any problem-specific information and the same hyper-heuristic could be run on all problem domains without change.

The HyFlex framework was used for a competition, the *Cross-domain Heuristic Search Challenge(CHeSC)*, in 2011, in which competitors submitted a single hyper-heuristic which was tested on 4 previously known domains along with 2 'hidden' domains. The strongest performing hyper-heuristic across the competition was that of Misir et al. [110]. The different elements of the algorithm are described below.

- **Dynamic Heuristic Sets** As with the method of [109], this algorithm maintains dynamic subsets of low-level heuristics for different phases of the search. In order to determine which heuristics are in the usable set, the low-level heuristics are first given a value for a performance measurement metric p_i. The p_i metric uses the following forms of feedback to calculate a value for heuristic i.
    - A measure of the number of best solutions found by this heuristic.
    - A measure of the total solution improvement over the search.
    - A measure of the total deterioration of solutions over the search.
    - A measure of the total solution improvement during the current phase.
    - A measure of the total solution deterioration during the current phase.

  Weights are assigned for each of the above measures, with the value of the weights being in a decreasing order for the order of measure above. Using this p_i performance measure, a quality index QI for a heuristic is generated according to a normalised ranking of p_i values. All heuristic with a QI value less than that of the average of the full set of heuristics is excluded from the subset for that phase. The authors of [110] call the length of time that the heuristic is excluded from the set the *tabu duration*. This is hardcoded for the [110] method. Similarly, a pre-determined constant value is used to determine the length of a phase.

- **Selection of Heuristics** Probabilities for heuristics within the dynamic set are determined as normalised values of the number of best improvements found by a solution with respect to time taken.

- **Relay Hybridisation** Relay hybridisation is also used within the Misir algorithm to select low-level heuristics. It is concerned with the performance of pairs of heuristics when applied consecutively. Firstly, a random variable determines the probability of using the relay hybridisation selection mechanism. Then, a first heuristic is selected by a *learning automaton* method that learns a heuristic's performance. Then a second heuristic is selected from a list of possible heuristic to follow the first heuristic. There is a separate list maintained for each heuristic with a fixed size. Variables keep track of the performance of the relay hybridisation selection method and for the dynamic heuristic set method above.

- **HyFlex Parameter Adaptation** HyFlex contains 2 parameters, the values of which can be specified by the user at any point of the search, *intensityOfMutation* and *depthOfSearch*. These parameters are used to control the strength of certain low-level heuristics. In [110], their performance is monitored by Misir et al. and their values adapted during the search based on this performance.

- **Solution Acceptance Method** The *adaptive iteration limited list-based threshold accepting with a fixed limit (AILLA-F)* method described above for [108] is used again here for solution acceptance.

- **Re-initialisation of Heuristic** A solution is re-initialised if a certain threshold has been reached in terms of iteration numbers without an improvement in objective function value. This is done with the aim of promoting diversification within the search.

The nature of the combination of many adaptive elements seems to be the key to the success of the algorithm of Misir et al. [110]. In the CHeSC competition's points-based scoring system, the method achieved around 35% more total points than its nearest competitor.

The second place hyper-heuristic was that of Hsiao et al., described in [82]. Their method is a *Variable Neighbourhood Search(VNS)* approach where two main stages of 'shaking' and local search are iterated between. The shaking stage aims to disrupt a solution and allow for some diversification. The authors extend the VNS method by adaptively modifying the length of the local search stage depending on the state of the search. The population size is also determined dynamically.

A hyper-heuristic based on an analogy of 'Pearl Hunting' is present by Chan et al. in [32]. The pearl hunting method describes a process of diversification and intensification where a hunter would dive (intensification) to find better solutions before re-surfacing and moving to a different area (diversification). Two levels of local search intensity are used in the algorithm and are controlled by manipulation of HyFlex's *depthOfSearch* parameter. The level of intensification is also controlled by only allowing 'deep diving' or heavy intensification for the best of the solutions found. Furthermore, the level of diversification is adaptively determined through a measure of whether or not the method is stuck in a local optima. The pearl hunter algorithm placed 4th in the CHeSC competition. Interestingly, it placed 1st in the 2 hidden domains, implying that it is able to adapt to new problems.

In [123], Özcan and Khieri present a multi-stage hyper-heuristic for use in the HyFlex framework. Initially, a greedy local search stage is applied to allow for data to be gathered regarding the performance of heuristics. This is then used to maintain an active heuristic set, which contains heuristics with the potential to perform well. A heuristic dominance method is then used to maintain this list, where heuristics that are dominated by other heuristics are excluded from the list. For the dominance calculation, the magnitude of an improvement and the number of steps required to find the improvement are considered. The method beats a set of basic hyper-heuristics when tested on the 4 problem domains of HyFlex.

A hyper-heuristic approach based on the choice function is proposed by Drake et al. in [54]. This paper extends the method of Cowling et al. [41] by introducing a technique to automatically determine parameter value used in the choice function to control the importance of different performance measures. Drake et al. observe that the original choice function can often suffer from too much diversification when the search gets stuck. To counter this, a greater emphasis on intensification is used in [54], with significant rewards attributed for improvements in objective function value. This method is implemented and tested in the HyFlex framework and, on the 4 problem domains, returns substantially better results than the original choice function.

2.4 Conclusion

The purpose of this chapter has been to identify research questions that can motivate the work that is to be presented in the remainder of this thesis. In this literature review, two significant areas have been considered. The first is the Vehicle Routing Problem, for which a problem description has been provided. As well as detailing early work on the problem, significant contributions to the field have been discussed in detail. These contributions come from a range of areas and utilise many differing techniques. Specifically, the variant of the problem known as the Vehicle Routing Problem with Time Windows has been the subject of most focus, both in the literature and consequentially in this review.

The second area of focus within this review has been that of adaptive search techniques called Hyper-heuristics. These methods, which operate on the heuristic space, rather than the solution space, aim to adapt to multiple problems and problem variants. The relatively recent origins of hyper-heuristics are described along with a review of work performed so far in this area.

Chapter 3

Vehicle Routing Problem Domain and HyFlex Framework

3.1 Introduction

As is clear from the title and introduction to this thesis, this work is concerned with the design of Vehicle Routing Problem domains for hyper-heuristics. The high-level question of how to design domains for hyper-heuristics can be broken down into several other research questions, as stated in the introductory chapter. One such question is 'What is a problem domain?'. This chapter will set out to define a basic, but flexible, definition of a problem domain which can be used to represent a wide range of problems and support a wide range of algorithms.

In addition to having a definition of a problem domain, it is important to understand what makes a 'good' problem domain. While this is a more subjective question than the first, this chapter will aim to provide an answer by considering in what way the different elements of a domain (as set out in the definition) impact upon the operation of an algorithm. Specifically, the relationship between the components of a domain and the workings of a hyper-heuristic will be explored and analysed. In order to further clarify the points raised, a Vehicle Routing Problem with Time Windows (VRPTW) domain will be presented, with design decisions for each domain component being discussed.

3.2 Problem Domain Definition

For a hyper-heuristic or adaptive algorithm to produce solutions to a problem, it could be argued that 2 things are needed. The first would be a representation of the problem to be solved. This representation may include the data structure and some instances of the problem. The second aspect needed would then be a set of tools to manipulate and operate on a solution that uses the representation of the problem. Included in these tools could be heuristics to modify a solution and objective functions to evaluate a solution. In the previous chapter, some frameworks for hyper-heuristics were discussed. One of these, HyFlex, has the advantage of existing problem domains, as well as a flexibility to implement a wide variety of algorithms and represent many problems. A fuller description of HyFlex will be given in the following section. However, it is important to mention it here as it is from the HyFlex composition of problem domain elements that the definition for this work is to be derived. Below is a brief summary of the components that make up a problem domain, split into the sections of problem representation and domain tools. A more detailed analysis of the relationship between these problem elements and the algorithms that use them will follow later in this chapter.

3.2.1 Problem Representation

Base Representation (Data Structure)

This component is the basic representation of the problem. It should contain sufficient information about the problem and inter-relationships within the problem for a solution to the problem to be modelled. As an example for the vehicle routing problem, this representation would be a set of routes. Each route would be an ordered set of customers. If the implementation of this component is being considered, the data structure to be used will also form part of the definition.

Constraints

The 'Constraints' component represents a set of specifications that must be met for a solution to the problem to be valid. The constraints should be defined with reference to the base representation of the domain.

Instances

As well as the ability to represent a problem, a domain also requires data (or instances) for the problem. These instances must agree with the defined representation of the problem, or be considered sub-problems that can be represented in the domain.

3.2.2 Domain Tools

Objective Function

In order for an algorithm to evaluate the success of its operations within a problem domain, it needs to have some means to measure the quality of a solution. Thus, each domain should have one or more objective functions which give a representation of the 'quality' of a solution. This 'quality' can be measured in different ways and it is the objective function which gives the definition of a high or low quality solution for a particular problem. It is arguable that this should fall into the 'Representation' section of the definition. However, the view taken here is that the objective function is a tool used by the algorithm that operates on the problem representation.

Low-level Heuristics

For an algorithm to be able to modify and potentially improve upon a solution, the domain must provide the means to manipulate the represented data. In this definition, we term these methods 'Low Level Heuristics'. Each of these make modifications to the data and should provide scope to the hyper-heuristic to modify a solution in different ways.

3.3 Discussion of Domain Components

Now that a basic definition of a problem domain has been established, the focus in this chapter turns to a more detailed discussion of the separate components of a domain. For each component, a number of factors will be considered to determine what makes that component suitable for usage by a hyper-heuristic or other adaptive algorithm. Firstly, a brief overview will be given of the HyFlex framework, which is to be used for problem domain design within this thesis. Following this, each of the 5 problem domain components described in the definition above will be discussed in more detail.

3.3.1 HyFlex Framework

HyFlex (Hyper-heuristic Flexible Framework) is a software framework, implemented in Java, which facilitates the development of hyper-heuristics and other general purpose algorithms. As has been shown in Chapter 2, there has been an increasing amount of research into algorithms and areas that can produce results that are less specialised. This generalisation can take many forms. An algorithm could be considered more general or adaptive if it works effectively on a variety of instances of a problem. Alternatively, generality can consider approaches that operate on a number of completely different problem domains, producing consistent results with few manual changes having to be made in between runs. You can further consider the branch of algorithms that either generate algorithms to solve specific problems, or that auto-tune parameters for effectiveness on different instances. A problem often arising when conducting work in this area of research has been the need for a vast number of instances with which to test algorithms. Whilst a researcher will often be an expert in a particular problem domain, and be familiar with a number of other problems, they will not necessarily have the sufficient level of knowledge of all of these domains that they would need in order to implement them for testing of their algorithm. Even for the domains in which they have specialist knowledge, it can still be time consuming to gather a sufficient number of instances with which to conduct comprehensive tests. The implementation of these extra domains, and the gathering of instances, can result in time being unnecessarily used when it could be put towards the further development of algorithms. It would appear that HyFlex provides a solution to these issues and will provide a strong base for the work presented in this thesis.

HyFlex [116] is based on the idea of a domain barrier separating problem specific details from the design of algorithms, as presented in [41]. Due to this separation, HyFlex can be considered as having two elements; the hyper-heuristic or adaptive algorithm element, and the problem domain element. The concept is given a pictoral representation in Figure 3.1. These will be separately described below.

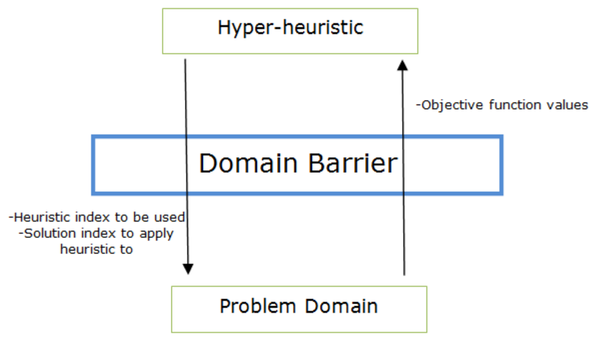

Hyper-heuristic
-Objective function values
Domain Barrier
-Heuristic index to be used
-Solution index to apply heuristic to
Problem Domain

Figure 3.1: A representation of the domain barrier present in HyFlex.

3.3.2 Problem Domain Design

The problem domain element of HyFlex has several parts to it. Firstly, there is an interface which designers of the problem domains can use to produce a domain for HyFlex. One element of a problem domain identified in the definition given above is the representation, or data structure, used to model the problem.

Design of Base Representation of Problem

There are several factors that a domain designer should consider when choosing a data structure to represent a problem. One such factor is flexibility. If a domain is being designed for use by hyper-heuristics, then it is likely that multiple low-level heuristics will be repeatedly modifying one, or several, solutions. Thus, a representation must allow for easy access to data elements and simple means to manipulate the ordering of these elements. Linked to this first point is the speed with which a data structure can be manipulated. Speed is an important factor as, by allowing more applications of heuristics within a given time, there is greater potential for improvement in solution quality. Finally, a data structure should be easily understandable and usable. In order to be kept up-to-date with the best performing algorithms, it should be simple for a domain designer to add new low-level heuristics to an existing domain. A simple representation of a problem will make this onboarding process easier and encourage more improvements to existing domains.

Management of Constraints in Problem Domain Design

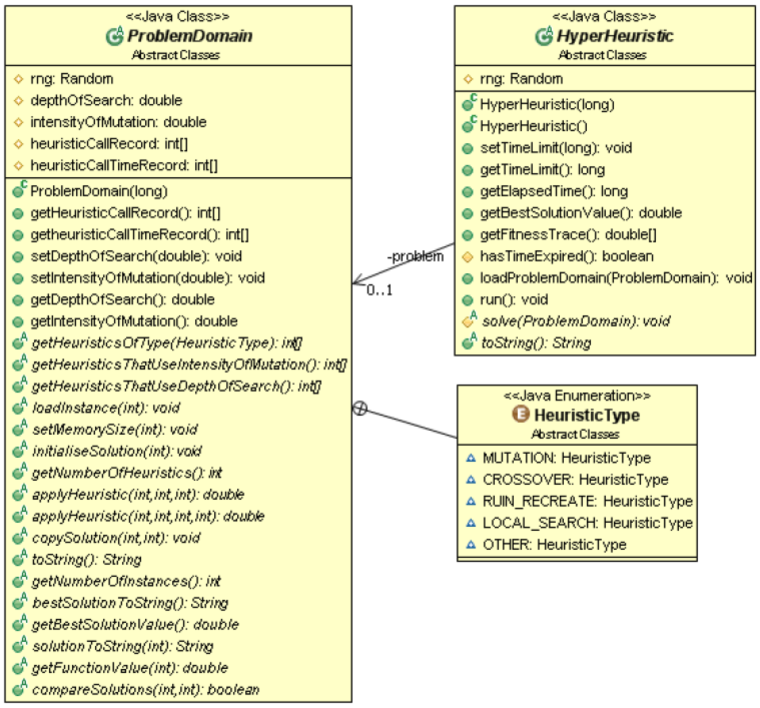
The choice, and enforcement, of constraints for a problem are inevitably linked to the base representation of a problem. The choice of data structure can determine whether it is simpler for constraints to be enforced either through a penalty function in the objective function or through the workings of the low-level heuristics. As an example, for a hard constraint, the domain could be designed in such a way that a low-level heuristic would never make a modification to a solution that would result in that constraint being violated. Alternatively, the domain could allow constraint violations, but with a high penalty in the objective function value. One advantage of this latter mechanism is that the solution could enter the infeasible search space - a situation which is deliberately used by some other algorithms to diversify the search. Another consideration of constraint management is whether constraints should be configurable for different instances of a problem. For example, an instance file could specify which constraints should apply for that instance. A downside to this approach is that it could be more complex to implement the domain than if there were a constant set of constraints. However, the increase in flexibility, and the ability to represent a wider range of problems, might make this extra effort worthwhile.

Choice of Low-level Heuristics

The problem domain designer is also responsible for the choice of low-level heuristics for the problem, as well as auxiliary methods to create and copy solutions. In making the selection of low-level heuristics, there should be a focus on ensuring that state-of-the-art heuristics are available to the hyper-heuristic. Through offering these, the algorithm has a greater opportunity to find high quality solutions. In addition, the domain should look to offer a sufficient number of low-level heuristics. In the context of a problem domain for hyper-heuristics, this means that there are enough heuristics to provide meaningful information to the hyper-heuristic when it is making intelligent decisions about heuristic selection. By constrast, the domain should not offer such a surplus of heuristics that the hyper-heuristic does not have enough time to deduce their relative performance. The low-level heuristics are split into four categories;

- Mutation Heuristics: Heuristics which make a small perturbation to a solution. An example of this might be a swap or insert heuristic.
- Local Search Heuristics: These heuristics produce a non-deteriorating solution by repeatedly applying a low-level heuristic. This low-level heuristic may be one of the mutation heuristic that is applied in the context of a best-improvement or first-improvement hill-climbing algorithm.
- Ruin-recreate Heuristics: These are heuristics which will destroy or mutate part or all of a solution before attempting to re-construct the solution. There is potential to use the other categories of low-level heuristic within this. For example, using a mutation heuristic for the ruin element of the heuristic, then using a local search or constructive heuristic to re-build the solution.
- Crossover Heuristics: These heuristics take two solutions and combine them to form a third solution.

Full class diagrams are given for HyFlex in Figure 3.2. The software interface allows the algorithm designer to see which heuristic indices belong to which of these categories although, as previously stated, the actual names and details of these heuristics are kept hidden. As can be seen in the class diagrams, the method *getHeuristicsOfType(HeuristicType)* makes it easy to identify heuristic categories. Several algorithms have structures where this knowledge of heuristic 'type' becomes crucial. For example, an evolutionary algorithm has multiple stages involving different sorts of low-level heuristics, such as mutation and crossover. HyFlex allows these branches of algorithms to be implemented, whilst maintaining the



<<Java Class>>
ProblemDomain
Abstract Classes
rng: Random
depthOfSearch: double
intensityOfMutation: double
heuristicCallRecord: int[]
heuristicCallTimeRecord: int[]
ProblemDomain(long)
getHeuristicCallRecord(): int[]
getheuristicCallTimeRecord(): int[]
setDepthOfSearch(double): void
setIntensityOfMutation(double): void
getDepthOfSearch(): double
getIntensityOfMutation(): double
getHeuristicsOfType(HeuristicType): int[]
getHeuristicsThatUseIntensityOfMutation(): int[]
getHeuristicsThatUseDepthOfSearch(): int[]
loadInstance(int): void
setMemorySize(int): void
initialiseSolution(int): void
getNumberOfHeuristics(): int
applyHeuristic(int,int,int): double
applyHeuristic(int,int,int,int): double
copySolution(int,int): void
toString(): String
getNumberOfInstances(): int
bestSolutionToString(): String
getBestSolutionValue(): double
solutionToString(int): String
getFunctionValue(int): double
compareSolutions(int,int): boolean
-problem
0..1
<<Java Class>>
HyperHeuristic
Abstract Classes
rng: Random
HyperHeuristic(long)
HyperHeuristic()
setTimeLimit(long): void
getTimeLimit(): long
getElapsedTime(): long
getBestSolutionValue(): double
getFitnessTrace(): double[]
hasTimeExpired(): boolean
loadProblemDomain(ProblemDomain): void
run(): void
solve(ProblemDomain): void
toString(): String
<<Java Enumeration>>
HeuristicType
Abstract Classes
MUTATION: HeuristicType
CROSSOVER: HeuristicType
RUIN_RECREATE: HeuristicType
LOCAL_SEARCH: HeuristicType
OTHER: HeuristicType

Figure 3.2: Class diagrams for the HyFlex framework.

important concept of a domain barrier.

Objective Function Design

The problem domain designer also has responsibility for calculation of the objective function, and for storage of the current best objective value, as obtained throughout the current search. This function must represent a minimisation problem, which ensures consistency and ease-of-use across all HyFlex domains. As only a single objective value is returned for each solution, there is a potential problem in representing multi-objective problems. A practical approach to this problem is to use a weighted sum to calculate the objective value. However, it is important that, firstly, it is made clear to the algorithm designer how the objective function is constructed so that they can make informed decisions on the actual value of their algorithms and, secondly, that where possible the composition of the objective value mirrors that of the literature standard for best known solutions. Many designers of algorithms will want to know not only how they perform in relation to other HyFlex algorithms, but also to the very best results in the literature. After all, leading the field of HyFlex algorithms is meaningless if the solutions produced are of a poor quality relative to solutions produced by specialist algorithms. What could be deemed a sufficient quality of solution is a matter for the hyper-heuristic or meta-heuristic designer; however, it is important that the tools exist for the necessary comparisons to be made.

Choice of Problem Instances

A further element of designing a problem domain for HyFlex is the choice of instances to be made available. Again, there are factors within this area to consider, if a HyFlex domain is to be a useful research tool. Where they exist, a domain should contain the most widely-studied or 'benchmark' problem sets for its problem area. Results for these instances are necessary to provide justification that an algorithm is performing well, as they will often be widely studied and have a high quality of best-known solution. The ability to attain strong results for these instances will be an indicator of an algorithm that works well within the respective problem domain. On the other hand, if a domain is to serve its purpose in relation to the philosophy behind HyFlex (that of allowing studies in generality and development of algorithms that will work across multiple instances and multiple problems), then there is a need to provide a range of instances for each problem domain. Of particular interest could be instances with additional constraints or a differing structure that might more closely represent real-world problems. To perform well on one set of instances shows only that an algorithm works well for that particular representation of the problem. Should that algorithm also then perform strongly on a new set of instances, particularly one that is more constrained or of a completely differing structure, without the need for any manual tuning, then there are indications that the algorithm can show adaptability. The downside to some 'real-world' type instances is that there is a lack of previous work on them, meaning that strong results on these instances can carry less weight than on more established instances. Instances are selected in HyFlex through the use of indices. A user will specify the index of the instance they wish to use and the problem domain will load this instance using the *loadInstance(int)* method (see Figure 3.2).

3.3.3 HyFlex Algorithm Design

Designers of algorithms for HyFlex are provided with a software interface to use to run their algorithm. Although, as mentioned above, problem-specific information about the domain is not obtainable through this interface, there still exist sufficient means for the designer to manipulate the search space and implement a wide selection of adaptive algorithms. These means relate to the tools given in the definition of a problem domain earlier in this chapter. This section will consider how the provision of tools can be used by different hyper-heuristics and what qualities are desirable for these tools to possess.

One such element of control is in the determination of population size for the search. The *setMemorySize(int)* (see Figure 3.2) method is used to set the initial size, which can be altered at any point during the search. Methods also exist to manage the population of solution, specifically by being able to copy solutions from one index to another (*copySolution(int,int)* where the ints represent solution indices) and through a method allowing a comparison between two solutions to indicate their equality (*compareSolutions(int,int*).

Once this population has been established, it can then be operated on in several ways. In addition to the methods stated above, a constructive or initialisation heuristic can be applied to a chosen solution. This is performed through method *initialiseSolution(int)* where the int value is the index of the solution to be initialised. This method can be seen in the class diagrams of Figure 3.2. Following this, two methods may be called to apply any of the set of low-level heuristics to any solution within the population. The first of these methods, *applyHeuristic(int,int,int)*, takes as input a heuristic index, a source solution index and a destination solution index. The heuristic with the specified index is then applied to the source solution with the resulting new solution being placed at the position of the destination solution's index. The second of the methods, *applyHeuristic(int,int,int,int)* takes as input a heuristic index, two indices for source solutions and a destination solution index. This method is designed for use mainly with crossover heuristics, which mostly require two parent solutions.

Crucially, the exact nature of the low-level heuristic is unknown to the algorithm, the only information available is the category of heuristics under which the heuristic is classified (mutation, ruin-recreate, etc.) and an index associated with this heuristic. By not revealing exactly which heuristic has been applied, it is ensured that successful usage of the low-level heuristics can only be attributed to sophisticated adaptive learning techniques and not through prior domain knowledge. Associated with the application of these low-level heuristics are two user-determined parameters that subtly affect their running. In the category of Local Search heuristics, there is a parameter named 'Depth of Search' with the variable name *depthOfSearch* that some heuristics may use to determine various aspects of the search. Similarly, the parameter 'Intensity of Mutation' with the variable name *intensityOfMutation* affects heuristics in the 'Mutation' and 'Ruin-recreate' categories. The values of the *depthOfSearch* and *intensityOfMutation* parameter can be modified using the *setDepthOfSearch(double)* and *setIntensityOfMu-*

tation(double) methods respectively. These are shown in the class diagrams in Figure 3.2. The values for these parameters must be between 0 and 1. The domain provides information concerning which heuristics use which parameters, but does not state exactly how they are used.

Further to the application of heuristics to the set of solutions, it is possible to determine an objective function value for each solution, as well as determining the lowest objective function value obtained so far during the search. The objective function value for a particular solution can be accessed through the *getFunctionValue(int)* method, where the int represents the index of the solution for which the objective function is requested. There is also a method, *getBestSolutionValue()* which returns the best objective function value found so far in the search. Only a single value is available for each solution, meaning that if the objective function is a weighted sum or some other multi-component function then the algorithm will not know which part of the magnitude of the value can be assigned to which component of the objective function.

As well as management of the population of solutions and application of low-level heuristics, the designer also has responsibility for the details of the running of the algorithm. There are multiple elements to be considered here. Firstly, an instance must be chosen for the algorithm to run on. A set of instances are provided with each problem domain and can be loaded through their index. As has been previously stated, the method *loadInstance(int* is used for this purpose. The user can also set a time limit for the run (using the *setTimeLimit(long* method of the HyperHeuristic class), as well as a seed value for use with any pseudo-random elements within the domain. These elements make it simple for the user to run multiple tests on several instances and with different seed values.

3.3.4 Pre-existing HyFlex Domains

There are four problem domains provided in the initial version of HyFlex. These are Permutation Flow Shop, Personnel Scheduling, One-dimensional Bin Packing and the Maximum Satisfiability Problem. The objective functions for all domains are minimisation problems. These all conform with the framework as described above and are shown in more detail below.

Permutation Flow Shop

Problem Description The Permutation Flow Shop problem domain represents the problem of ordering a set of n jobs which have to be processed on a set of m machines in a set sequence. Each job has a processing time associated with each machine and the sequence of jobs for the first machine must be maintained across all subsequent machines. In other words, one job cannot overtake another job. There is no unnecessary waiting time permitted. That is to say if a machine is free and there is a job waiting to be processed, it must be processed immediately. A technical report for this domain can be found in [167].

Objective Function The objective function is to minimise the overall time for all jobs to be completed, in other words the makespan.

Low-level Heuristics To initialise solutions, a randomised version of the NEH algorithm [113] is used. For the other heuristic categories, there are 5 mutation, 2 ruin-recreate, 4 local search and 3 crossover heuristics totalling 14 low-level heuristics. The crossover heuristics are based upon some standard heuristics for permutation representations and the ruin-recreate heuristics use the same NEH method as initialisation during the 'recreate' section of the method. Inspiration for the mutation and local search heuristics comes from [140] and [141].

Instances Instances for this domain have been taken from the benchmark Taillard set [152]. These are instances of differing numbers of jobs and machines. The processing times for jobs have been generated randomly within a specified interval.

Personnel Scheduling

Problem Description Personnel Scheduling can refer to a wide range of problems with many constraints and characteristics. In its simplest form, it can be described as assigning workers to shifts over a set time period. This HyFlex problem domain makes use of a specialist instance file in order to allow representation of the wide range of problems that fall under the personnel scheduling umbrella. A technical report for this domain can be found at [43].

Objective Function Due to the changeable nature of the problem, different instances will have different objective functions, which will be specified in the instance file. Constraints also form an element of the objective function, with any violations being highly penalised within the objective function.

Low-level Heuristics A simple local search heuristic is used to initialise the solution. Over the other categories, there are 1 mutation, 3 ruin-recreate, 5 local search and 3 crossover heuristics totalling 12 low-level heuristics. Many of these heuristics are inspired by the literature for nurse rostering problems, specifically [19], [24], [25] and [26].

Instances The instances for the personnel scheduling domain combine instance both from the academic community and from real-world problems with a wide range of constraints and objectives. They are all taken from the 'Staff Rostering Benchmark Data Sets' [44].

One-dimensional Bin Packing

Problem Description The One-dimensional Bin Packing Problem is that of placing a number of differently sized items into a number of bins. The size of an item is represented as an integer and each bin has a capacity, also an integer value. A technical report for this problem domain can be found at [86].

Objective Function The general objective of bin-packing is to minimise the number of bins used. For this domain, the objective function measures fullness of bins in order to drive the search toward reducing the overall number used.

Low-level Heuristics A randomised version of the 'first-fit' heuristic [88] is used to initialise solutions. For the other heuristic categories, there are 2 mutation, 2 ruin-recreate, 2 local search and 1 crossover heuristic, totalling 7 heuristics across all categories.

Instances The instances are taken from a number of sources and represent benchmarks for the one-dimensional bin packing problem. Instances come from sources including [41] and [8].

Maximum Satisfiability (MAX-SAT)

Problem Description The Maximum Satisfiability problem is that of satisfying as many clauses as possible within a boolean logic formula. To satisfy a clause means to assign truth values to the variables within a clause in such a way that the clause as a whole evaluates to true. A technical report for this problem domain can be found at [85].

Objective Function The objective function is to minimise the number of unsatisfied clauses within the formula.

Low-level Heuristics Solution initialisation is done in a uniformly random manner, with each variable being assigned either a true or false value. For the other heuristic categories, there are 2 mutation, 1 ruin-recreate, 4 local search and 2 crossover heuristics, totalling 9 heuristics across all categories. The mutation heuristics are simple bit-flip operations, with multiple variables to be 'flipped' chosen at random. Standard crossover methods such as 1-point and 2-point crossover are used. For the local search heuristics, a number of advanced methods from the literature are used, including [67], [149] and [148].

Instances The instances are taken from benchmark sets at [81].

3.4 VRPTW Domain

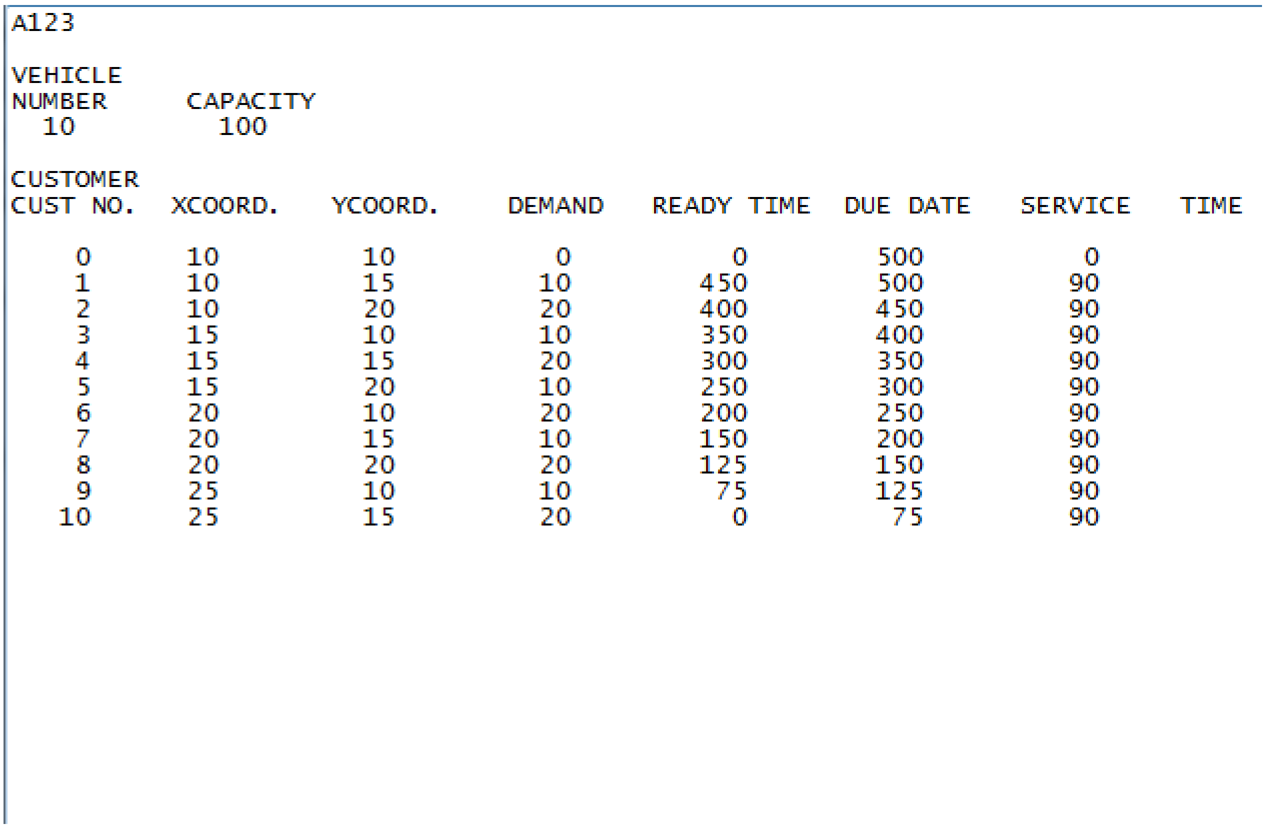
In the previous sections, a definition of a problem domain has been given along with an analysis of the qualities needed for a domain to be 'useful' to hyper-heuristics. These concepts shall now be demonstrated further through the presentation of a new HyFlex domain for the Vehicle Routing Problem with Time Windows (VRPTW). As this thesis is concerned with the design of VRPTW problem domains for hyper-heuristics, this domain will allow further experimentation and discussion about the relationship between a hyper-heuristic and vehicle routing problem domains. A full description of the VRPTW has been given in the previous chapter. However, in short, it is the problem of satisfying a set of customer demands, by creating routes that begin and end at a depot location. Each route can be seen as a permutation of customers. These routes (or vehicles) are subject to capacity constraints and each customer has a start and end time-point in between which their demand must be met. The primary objective of this problem is to minimise the number of vehicles needed with a secondary objective of minimising the total distance travelled. Presented here will be a problem domain for the VRPTW, implemented within the HyFlex software framework. As they have been described above in general terms, so will the various aspects of a problem domain be described here in terms specific to this VRPTW problem domain.

3.4.1 Problem Representation

In general terms, the VRPTW problem has been implemented here as a list(array) of *Route* objects, each of which represents a doubly-linked-list, beginning and ending with a depot location. The routes are stored as a list as they are not ordered and as such have no need for a more complex data structure. It is thought that a doubly-linked-list is ideal for a representation of individual routes as manipulation of the solutions within a VRPTW domain will require tracking back through a route to correct waiting times. The return link provided by the double linked list makes this process more efficient. The choice of data structures is important as the operators will be modifying the data so frequently that small improvements in efficiency will multiply and can contribute to a reduction in processing time for each operation and, hence, the possibility for an increase in solution quality as it will be possible to perform a greater number of iterations within a set time limit.

3.4.2 Instance Format and Instance Sets Used

The instance format used for this domain is the standard format for VRPTW problems. The files have three main elements, which can be explained as follows. It is important to note that the values of distance and time are the same. As an example, to travel a distance of 67 would take 67 units of time. An example VRPTW instance file can also be seen in Figure 3.3.



```
A123

VEHICLE
NUMBER     CAPACITY
  10         100

CUSTOMER
CUST NO.   XCOORD.    YCOORD.     DEMAND    READY TIME   DUE DATE    SERVICE    TIME

    0        10         10          0           0          500          0
    1        10         15         10         450          500         90
    2        10         20         20         400          450         90
    3        15         10         10         350          400         90
    4        15         15         20         300          350         90
    5        15         20         10         250          300         90
    6        20         10         20         200          250         90
    7        20         15         10         150          200         90
    8        20         20         20         125          150         90
    9        25         10         10          75          125         90
   10        25         15         20           0           75         90
```

Figure 3.3: A VRPTW instance file

Name of Instance A single string value giving the name of the instance.

Vehicle Constraints Two values regarding constraints for the vehicles/routes.

- *No. of Vehicles* An integer for the maximum number of vehicles to be used for the instance.
- *Vehicle Capacity* An integer representing the maximum capacity of each vehicle.

Customer Data A number of values regarding the customers in the instance.

- *Customer No.* An integer representing an index for that customer. Customer 0 is always the depot location.
- *X Coordinate* An integer value for the location of the customer on the x axis.
- *Y Coordinate* An integer value for the location of the customer on the y axis.
- *Demand* An integer value representing the specific demand for that customer.
- *Ready Time* An integer value representing the earliest time a customer can be serviced.
- *Due Date* An integer value representing the latest time a customer can begin to be serviced.
- *Service Time* An integer value representing the amount of time taken to service that particular customer.

The instances for this domain have been taken from two sources. The first are Solomon's 56 benchmark instances [151]. Each of these instances contains 100 customers. The second data set is larger, with 1000 customers in each instance. This is the Gehring and Homberger dataset, also of 56 instances [79]. For each set of instances, there are three categories of instance. The first, **C**, have subsets of customers which are clustered together in terms of location. The second, **R**, has the customers dispersed randomly and the third, **RC** has a combination of the two. These instance sets have been chosen as they have been widely used and tested for many VRPTW approaches. For an algorithm to prove its worth, it has to show strong results for these instances so their inclusion is necessary in order for this to be a useful domain. According to the analysis earlier in the chapter, a domain should have a range of instances that represent sufficiently different permutations of a problem. With the different categories of instances demonstrated in these datasets, as well as the different sizes of instances, these datasets would seem to be a suitable initial choice.

3.4.3 Initialisation of Solution

It is a requirement of HyFlex problem domains that the solution initialisation process is stochastic, in order to ensure that population-based algorithms can have a diverse initial population with which to begin their search. The requirement to have a stochastic process means that the method to be used differs from the most popular literature methods. However, there are also other advantages to this method, which are discussed following the description below. The pseudocode for this heuristic can be seen in Algorithm 1. Initially, the depot is extracted from the list of customers that has been built from the instance data. A single empty route, *r*, is generated as the initial solution. The method loops until the list of unrouted customers, *C*, is empty. At each iteration, it is first determined whether any of the currently unrouted customers could be inserted at the end of *r* whilst satisfying all constraints (the method *feasibleCustExistsForRoute(r,C)* is used in the pseudocode). If there is no feasible customer for *r*, the route is closed and a new empty route is created and the algorithm continues. If it's the case that there is a feasible customer, then the algorithm must determine which is the 'best' feasible customer for insertion. This element is performed by the method *selectBestCustomer(r,cLast)* in Algorithm 1. The method operates as follows:

- For each unrouted customer, it is first checked whether it can be feasibly inserted at the end of the route, *r*.
- For all feasible customers, a 'score' is calculated. This score contains three elements. The first, $dist_c$, is the Euclidean distance between the customer, *c*, and the current final customer in the route, *cLast*. The second, $time_c$, is the difference in the values of the *Ready Time* for *c* and *cLast*. The third is a random floating point number between 0 and 1.0, in order to maintain the required stochastic nature of the element.
- The complete formula for the score is: $score = (dist_c + time_c) * randNumber$
- The customer, *c*, with the lowest *score* value is selected.

Once the customer with the lowest score, customer *c*, has been selected, it is inserted at the end of route *r*. It is then removed from the list of unrouted customers, *C*. The process then repeats until this list is empty. The objective of this initialisation method is to provide a solution of a reasonable quality with which to start a search. However, it is desirable that the solution quality is not so high that a) the solution has already found a local minimum or minima and may be difficult to manipulate in such a way as to move toward the global minimum and b) that there is not sufficient diversity between solutions as they occupy similar areas within the search landscape. This is enforced through the inclusion of the stochastic element mentioned above.

Algorithm 1 The *Constructive Heuristic* algorithm for the VRPTW HyFlex domain which takes an instance *inst* as input.

```
procedure CONSTRUCTIVE HEURISTIC($inst$)
    List¡Customer¿ $C \leftarrow extractCustomers(inst)$
    Customer $depot \leftarrow C[0]$
    $C \leftarrow \{C[1]...C[n-1]\}$
    Route $r \leftarrow createEmptyRoute(depot)$
    List¡Route¿ $R \leftarrow r$
    while $size(C) > 0$ do
        if $feasibleCustExistsForRoute(r, C)$ then
            Customer $cLast \leftarrow getFinalCustomerInRoute(r)$
            Customer $c \leftarrow selectBestCustomer(r, cLast)$
            $r \leftarrow insertAtEndOfRoute(r, c)$
            $C \leftarrow removeFromList(C, c)$
        else
            $r \leftarrow createEmptyRoute(depot)$
            $R \leftarrow \{R : r\}$
        end if
    end while
    Solution $s \leftarrow createSolutionFromRoutes(R)$
end procedure
```

3.4.4 Low-level Heuristics

The twelve low-level heuristics for this domain are spread across the four categories defined by HyFlex, Mutation, Local Search, Ruin-Recreate and Crossover. They are mostly taken from proven methods in the literature, with the aim being to provide a domain that has the state-of-the-art tools needed to allow an algorithm to produce strong solutions. This provides a significant advantage to the problem domain and an interesting contribution to research as, although it is mostly existing methods being used here, the combination of these methods and their categorisation is new and has the potential to add value to the research area. In compliance with the HyFlex framework, all heuristics will return solutions that satisfy all hard constraints for the VRPTW, namely vehicle capacity constraints and time window constraints. For each of the above categories, a general overview will be given, along with detailed descriptions of the individual operators within that category.

Mutation Heuristics

A mutation heuristic is one that makes a small perturbation or alteration to a solution. A common reason for the usage of these heuristics is to help escape a local minimum in order to advance the search. They can also be seen as simple move operators which can be applied within user-defined acceptance criteria should they wish to use their own methods rather than apply the local search operators described in the following category. All of these heuristics will return any constraint-compliant solution, regardless of whether the objective function value has improved or deteriorated.

Two-opt This is a simplified version of the Two-opt heuristic described in [16]. It provides a minor modification to the solution by swapping two adjacent customers within a route. Note that, although it shares the same name, it is not the same method as the *2-opt* of Lin[98]. The Lin method consists of multiple moves in which 2 edges of a route are modified. The method proposed here is a simplification of the Lin method where the nodes in the edges to be swapped are adjacent to one-another. This heuristic has the potential to have an effect on the overall distance travelled, but not on the number of routes needed. The intensity of mutation parameter is used within this heuristic in order to determine how many swaps should be performed in a single application of the heuristic and hence to what extent the solution has been mutated. An example of Two-opt is given in Figure 3.4 and the pseudocode for the algorithm is at Algorithm 2. As can be seen the number of iterations is determined as the number of routes multiplied by the *intensityOfMutation* parameter value. Then, for each iteration, a route, r, is selected at random. From r, a customer, c_j, is selected at random (although the first customer in the route is excluded from selection). The next stage is to swap customer c_j with the customer preceding it in the route, customer $c_{j\text{-}1}$. The route is then checked for feasibility. If feasible, it is accepted into the solution and the algorithm continues.

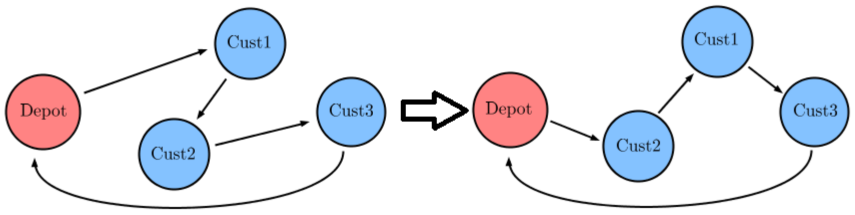


Figure 3.4: Example of two-opt. Customers 1 and 2 are swapped in this route.

Algorithm 2 The *Two-Opt* mutation algorithm takes as input a Solution s.

```
procedure TwoOpt($s$)
    for $i \leftarrow 0, intensityOfMutation * numOfRoutes$ do
        Route $r \leftarrow selectRandomRoute(s)$
        Route $r' \leftarrow r$
        Customer $c_j \leftarrow selectRandomCustomer(r')$
        $r' \leftarrow swapPositions(c_j, c_{j-1}, r')$
        if $feasible(r')$ then
            $r \leftarrow r'$
        end if
    end for
end procedure
```

Or-opt The Or-opt algorithm, first presented in [120], has similarities to the previously described two-opt heuristic. In this case, two sequential customers are selected in a single route, and moved to another location within that same route. This can be useful in providing a slightly more significant mutation than a single two-opt operation. Again, only the distance travelled is affected and the intensity of mutation parameter is used to determine the number of operations to be performed. An example of Or-opt is given in Figure 3.5 and the pseudocode in Algorithm 3. Again, the calculation of the number of routes multiplied by the *intensityOfMutation* value is used to calculate the number of iterations to be performed. At each iteration, a random route, r, is selected from which a random customer, c_j is selected (providing it isn't the final customer in the route). Customer c_j and the customer immediately following it, customer c_{j+1} are then removed from the route. A new customer, c_k is then selected at random from r and customers c_j and c_{j+1} are inserted immediately before c_k, whilst preserving their ordering. If the resulting route is feasible, it is kept and the algorithm moves on to its next iteration.

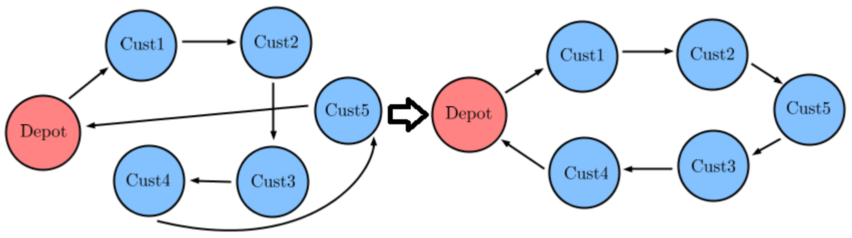


Figure 3.5: Example of or-opt. In this case, customers 3 and 4 are located so that they are serviced after customer 5, whilst preserving the ordering between them.

Shift The Shift heuristic, proposed in [143] differs from the two previous mutation heuristics in that it can alter the solution both in terms of the overall distance travelled and in terms of the number of routes/vehicles needed.

Algorithm 3 The *OrOpt* mutation heuristic takes as input a Solution s.

```
procedure OrOpt($s$)
    $timesToPerform \leftarrow (intensityOfMutation * numOfRoutes)$
    for $i \leftarrow 0, timesToPerform$ do
        Route $r \leftarrow selectRandomRoute(s)$
        Route $r' \leftarrow r$
        Customer $c_j \leftarrow selectRandomCustomer(r')$
        $r' \leftarrow removeCustFromRoute(r', c_j)$
        $r' \leftarrow removeCustFromRoute(r', c_{j+1})$
        Customer $c_k \leftarrow selectRandomCustomer(r')$
        $r' \leftarrow insertCustBeforeCust(c_j, c_k, r')$
        $r' \leftarrow insertCustBeforeCust(c_{j+1}, c_k, r)$
        if $feasible(r')$ then
            $r \leftarrow r'$
        end if
    end for
end procedure
```

The heuristic operates by relocating a single customer from one route to another. In this mutation version of the heuristic, customers and routes are chosen randomly, as the aim is simply to mutate the solution, rather than to directly improve it. The potential for a change in number of routes comes from the possibility of removing a customer from a route where that customer had been the only customer in the route. Thereby, its removal would cause the route to be empty and hence removed. There is also the chance for the number of routes to increase as if, once a customer has been removed, it can not be placed in another route in such a way as to satisfy all constraints, then a new route will be created into which the customer will be placed. The intensity of mutation parameter is used to determine the amount of operations to be performed. An example of Shift is shown in figure 3.6 with pseudocode for the method shown in Algorithm 4. Note that Algorithm 4 also makes use of a method named *insertCust*, which can be seen in Algorithm 5. The heuristic works as follows.

An important consideration when designing mutation heuristics is to ensure that a wide variety of moves are possible, and that the same customers don't continuously get moved. It is important so that diversity can be maintained within the search. For this reason, the initial choice of a customer to be removed is made randomly, from a random route. The insertion of this customer, however, tries to ensure that solution quality doesn't suffer to too great an extent. This is why the *bestInsertionPosition* method is used to find a strong insertion position for the removed customer. The measure used to determine this best insertion point is the required waiting time should the customer be inserted at that point. The position with the minimum waiting time is selected for insertion. This insertion should give a fairly strong solution but is not necessarily optimal, meaning that there is room for further optimisation.

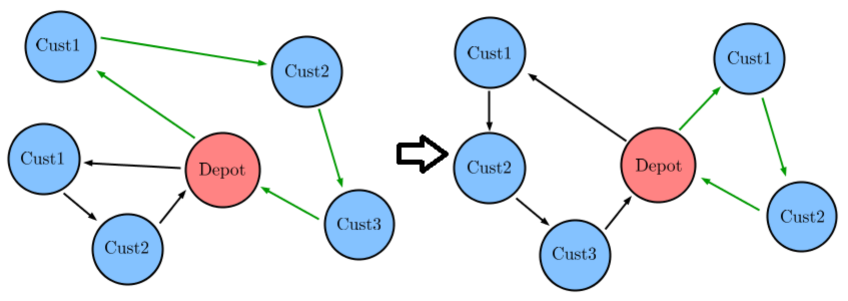


Figure 3.6: Example of shift. Customer 1 from the green route is relocated to become the first customer in the black route.

Algorithm 4 The *Shift* mutation algorithm takes as input a Solution s.

```
procedure SHIFT MUTATE($s$)
    for $i \leftarrow 0, intensityOfMutation * numOfRoutes$ do
        Route $r \leftarrow selectRandomRoute(s)$
        Customer $c \leftarrow selectRandomCustomer(r)$
        $r \leftarrow removeCustomerFromRoute(c, r)$
        $s \leftarrow insertCust(s, c)$        ▷ See Algorithm 18 for workings of insertion.
    end for
end procedure
```

Interchange Also proposed in the Savelsbergh paper [143], Interchange is a heuristic which performs a swap on two customers from separate routes. This heuristic can change the overall distance travelled, but not the number of routes in a solution. An example of the Interchange heuristic can be found in figure 3.7.

The pseudocode for this method is given in Algorithm 6. For each route, r, the number of times a move is to be performed is defined as the value of the *intensityOfMutation* parameter multiplied by the number of routes so that the level of mutation is proportional to the size of the problem. In the pseudocode, it is mentioned that the first customer, c^1, is selected but does not give the details as to how this occurs. The customer is selected through use of a metric that measures how 'out of place' each customer is within r. This metric, *score*, calculates the

Algorithm 5 The *insertCust(s,c)* method which takes as input a solution s and a customer c.

```
procedure INSERTCUST($s, c$)
    if $thereExistsFeasibleRoute(s, c)$ then
        Customer,route $c', r \leftarrow selectBestInsertionPosition(s, c)$
        $r \leftarrow insertBeforeCust(r, c, c')$
    else
        Routes $R \leftarrow getRoutes(s)$
        Route $r \leftarrow createNewRoute()$
        $r \leftarrow insertAtEnd(r, c)$
        $R \leftarrow \{R : r\}$
    end if
end procedure
```

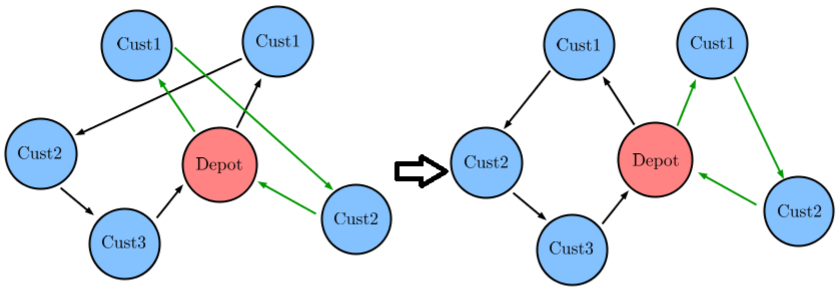


Figure 3.7: Example of interchange. Customer 1 from the green route is swapped with customer 1 from the black route. Correct ordering of all other customers is maintained.

difference between the due dates of the customer preceding *c*^{1} and the customer following *c*^{1}. Then the euclidean distance difference is calculated between these customers. This is added to the due date difference. Finally, the entire value is multiplied by a random number between 0 and 1 to ensure that different customers will be selected in different iterations of the algorithm.

The next element of the algorithm to be explained is how the 'swap' route, *r2* is to be selected. The aim is to find a route in which the customers are 'generally' close to the customer to be inserted. This is to be the route where the second customer will be selected. To select this route, a calculation is made for each route, *r'* in the solution. This calculation measures the average distance of all customers within *r'* to the previously chosen customer, *c*^{1}. The route with the lowest average distance is selected.

Following selection of the second route, *r2*, the second customer must be selected from within *r2*. The same closeness measure is used to select the second customer as was used to select the first customer. The customer in route *r2* that has the lowest sum of due date difference and euclidean distance from c^1 will be selected and swapped with c^1.

Algorithm 6 The *Interchange* algorithm takes as input a Solution *s*.

```
procedure INTERCHANGE($s$)
    for all Route $r$ in $s$ do
        $timesToPerform \leftarrow (numberOfRoutes * intensityOfMutation)$
        for $i \leftarrow 0, timesToPerform$ do
            Customer $c^1 \leftarrow selectFirstCustomer(r)$
            Route $r2 \leftarrow selectSwapRoute(c^1, s)$
            Customer $c^2 \leftarrow selectBestSwapCustomer(c^1, r, r2)$
            $r, r2 \leftarrow swapCustomers(c^1, r, c^2, r2)$
        end for
    end for
end procedure
```

Ruin-recreate Heuristics

The concept of a ruin-recreate heuristic was first proposed in [144] by Schrimpf et al. In that paper was also introduced the idea of *radial ruin* for the ruin stage of the method. Using this method, a number of customers are selected to be removed from the solution according to their proximity to a 'base' customer. The rationale behind this approach is to identify customers that are 'close' in terms of their value for certain properties. By removing and re-inserting these customer, an improved solution may be found without the need to substantially modify the state of a solution every time this heuristic is called. Two proximity measures are used here, those of euclidean distance and difference in time windows, which were proposed by Schrimpf et al. [144]. Those measure, as well as the random selection of a base customer, mirrors the method in [144]. The method here, though, differs in the selection of how many customers are to be removed from the solution. Schrimpf et al. select a number at random between 0 and the total number of customers. For this Hyflex method, customers are removed if their distance from the base customer falls within a pre-determined limit. The limit in this domain is chosen in such a way as to remove a large proportion, but not all, of a solution's customer when the *intensityOfMutation* parameter value is 1. If all customers were removed, it would be no different from re-initialising the solution. This limit is calculated using the following formula:

$$distanceLimit = intensityOfMutation * \tfrac{4}{5} furthestFromDepot$$

The value of $\frac{4}{5}$ is an arbitrarily chosen value which is designed to allow for a large amount of mutation when required by the *intensityOfMutation* parameter value. The value *furthestFromDepot* is the distance value between the depot of the base customer and the customer furthest away from that depot. Following the removal of customers from the solution, the recreate step commences. For this implementation of the algorithm, in this stage each removed customer is considered in the order they were removed and inserted back into the solution using the *insertCust(solution,customer)* method described in Algorithm 5 above. The pseudocode for this heuristic can be seen in Algorithm 7.

Algorithm 7 The *Location-based Radial Ruin* ruin-recreate algorithm takes as input a Solution *s*.

```
procedure LOCATIONRR($s$)
    Customer $baseC \leftarrow selectRandomCustomer(s)$
    $distLimit \leftarrow (intensityOfMutation * \frac{4}{5} furthestFromDepot(s, baseC))$
    $removedCusts \leftarrow baseC$
    for all Customer $c$ in $s$ do
        if $calcDistance(c, basec) > distLimit$ then
            $s \leftarrow removeCustFromSol(c, s)$
            $removedCusts \leftarrow c$
        end if
    end for
    for all Customer $c$ in $removedCusts$ do
        $s \leftarrow insertCust(s, c)$
    end for
end procedure
```

Below are presented the two version of ruin-recreate available within this domain.

Time-based Radial Ruin [144] This heuristic follows the method given above. For a proximity value, the difference between the current arrival times of the benchmark customer and the customer under consideration is used. Where this value is below the upper bound, which has been calculated as described above, then the customer is removed from the solution.

Location-based Radial Ruin [144] For this version of a ruin-recreate heuristic, the proximity value is calculated as the euclidean distance between the benchmark customer and the customer under consideration. All other aspects of the method are the same as above.

Local Search Heuristics

As with mutation heuristics, local search heuristics are concerned with making modifications to a solution, often in the form of swaps or insertions. However, unlike mutation heuristics which will return a solution regardless of whether the quality has improved or deteriorated, a local search heuristic will only return a solution whose quality is equal or better than it was to begin with. Also known as hill climbing heuristics, the methods presented in this category are a combination of basic process and solution acceptance. All of the heuristics presented here (except Two-opt*) follow a means of solution acceptance called 'first-improvement', whereby the operator will accept the first solution found where the objective function value is superior or equal to the previous un-modified solution. The alternative to this would be 'best-improvement' where all possible moves (or a certain subset of moves) would be considered before choosing the resulting solution which yielded the most significant improvement in objective function value. There were two drawbacks seen in this method; firstly that it was more costly to perform as a greater number of moves were often considered before a new solution was accepted, secondly that accepting the 'best' solution could often lead to getting stuck in a local optimum and be detrimental to the search as a whole. To clarify, the 'first-improvement' operators used here will iteratively perform basic operations, be that a swap or otherwise, and will immediately accept any move that has resulted in an objective function value equal or superior to that of the solution before the move. It is important to note here that moves which result in an equal objective function value are accepted, as well as improving moves. The decision to allow this is motivated by the benefit of so-called 'lateral' moves which can have a positive effect in diversifying the search by moving the search into a different area and potentially escaping local optimum. All of these operators make use of the 'depth of search' parameter in the same way. It is used to determine how many times the process of moves then acceptance will be repeated. In other words, it determines how many new solutions will be accepted. The four heuristics in this category are described in detail below.

Shift The basic mechanics of this heuristic are the same as that described in the mutation heuristic category, as originally presented in [143]. That is to say a customer is removed from one route and inserted into another route. However there are a number of difference between the mutation and local search versions of this heuristic. The key point for the local search version is that information from the solution is used to aid selection of the customer for removal. As before, the first stage of the algorithm is selection of a customer to be removed. For the mutation heuristic, this is performed in a uniformly random manner. Here, the starting route is still randomly chosen; however the customer to be removed is selected by a metric that, for each customer in the route, measures its distance between itself and the customers on either side. The metric also includes a small element of randomness achieved through use of the Java Random class to ensure that the same customer does not repeatedly get selected. The customer in the route that has the greatest value for this metric is determined to be most ‘out of place’ and is removed from the route. The aim behind this is to remove customers that are contributing most to the overall distance and hence hopefully lower the distance.

The method for selecting the insertion position for the removed customer is the same as in the mutation heuristic as that method is already geared toward solution improvement. Once a full move has been made, it is only accepted if it provides an improvement in objective function. This is in contrast to the mutation version, where the resulting solution is always accepted.

Interchange This heuristic works the same as the mutation version with the only difference being that a new solution is only accepted if it improves the objective function value. For the mutation version, the new solution is always accepted.

Two-opt* This heuristic, introduced in [130] by Potvin and Rousseau, operates by swapping the end sections of two routes to create two new routes. This powerful heuristic has the potential to improve the objective function both in terms of distance travelled and number of routes required. The first stage of this method is to select the two routes whose end sections are to be swapped. This is done in a uniformly random way to allow for diversity in the solutions created. The next stage is to select the start points for each route. In this context, the start point refers to a customer for which all customers following this customer will be placed at the end of the other route. The selection of start points for the routes could be considered as a ‘best-improvement’ process. All possible start points for the two routes are considered and then, once feasibility has been checked, are scored according to the new objective function value that would result from swapping at these points. This approach is used to give the highest quality solutions possible. The combination of points that would give the best objective function value is chosen and the swap is then performed. It was mentioned before that it is possible for this heuristic to alter the number of routes in the solution. The number of routes can be reduced in the situation where the start point of one route is selected as the final visit to the depot (i.e. the final stop in the route) and the start point of the other route is the first customer in that route. The depth of search parameter is utilised here to control the number of times this process is performed. Specifically, the process will be performed *depthOfSearch*numberOfRoutes* times so that the number of applications is proportional to the size of the solution. Pseudocode for this algorithm can be seen in Algorithm 8.

Algorithm 8 The *Two-opt** algorithm takes as input a Solution s.

```
procedure TWO-OPT*($s$)
    for $i \leftarrow 0, depthOfSearch * numOfRoutes$ do
        Solution $s' \leftarrow s$
        Route $r_1 \leftarrow selectRandomRoute(s)$
        Route $r_2 \leftarrow selectRandomRoute(s)$
        $bestScore \leftarrow getMaximumNumber()$
        Customer $best_c1 \leftarrow -1$
        Customer $best_c2 \leftarrow -1$
        for all Customer $c_1$ in $r_1$ do
            for all Customer $c_2$ in $r_2$ do
                $score = calcSwapScore(c_1, c_2, r_1, r_2)$
                if $score < bestScore$ then
                    $bestScore \leftarrow score$
                    $best_c1 \leftarrow c_1$
                    $best_c2 \leftarrow c_2$
                end if
            end for
        end for
        $s' \leftarrow performSwap(best_c1, best_c2, r_1, r_2)$    ▷ Perform swap of customers
following $c_1$ in $r_1$ with customers following $c_2$ iin $r_2$
        if $objFunc(s') > objFunc(s)$ then
            $s \leftarrow s'$
        end if
    end for
end procedure
```

GENI GENI, which was proposed in [65] by Gendreau et al., could be seen as similar to Shift in that it removes a customer from one route and relocates that customer to another route. Selection of the routes is performed randomly as with Shift. Selection of the customer to be removed is also similar, with a metric used to find the most 'out of place' customer. The metric is the same one that selects a customer in the above *shift* heuristic. Insertion of this customer into a new route is where the GENI heuristic differs from Shift. In Shift, a customer is placed into the best possible location in the route, with ordering between customers being preserved. For GENI, the customer is placed in between the two other customers in the route that it is closest to (determined using the same metric as used to select the customer to be removed) For purposes of clarity, these two customers shall be referred to as *c1* and *c2* and the customer to be inserted as *ic1.* It will sometimes be the case that the two closest customers are not currently sequential within the route. This will require a re-ordering of customers in the route. The customer to be inserted will be placed after *c1* or *c2*, whichever occurs earliest in the route sequence. For now, let's assume it is *c1.* All customers before *c1* will maintain the same ordering. Customer *c2* will be placed after customer *c1.* Following this all remaining customers in the route will be re-inserted into a position following customer *c2.* At each iteration of this stage, the customer with closest proximity (again determined by the same metric as above) is inserted at the end of the route. This process will be repeated until all customers have been reinserted into the route. Once this has been performed, the solution is checked for feasibility before being accepted if providing a superior or equal objective function score, or rejected otherwise.

Crossover Heuristics

A crossover heuristic in HyFlex is one that takes two solutions as input and returns a new solution derived from the inputs. This can often be done through methods such as One-point, Two-point and Uniform crossover where the problem has a simple binary representation. It would be fair to say that one main purpose of crossover heuristics is to combine the strongest qualities from the two parents in order to produce a superior child. This can either be forced, by having an operator that cherry-picks the best elements of a solution, or through chance, perhaps when a larger population is present and the expectation is that over time solution quality will be driven higher. Two new heuristics are proposed for this domain which between them cover both of these objectives. Each heuristic will be described below in greater detail.

Combine The Combine heuristic selects a combination of routes from 2 parent solutions to form a child solution. For this method, Combine determines a percentage value, x in order to control how many routes are included from the first solution. The percentage to be chosen, x is determined in a uniformly random manner but is between the values of 25% and 75%. These limits are selected arbitrarily to allow for a vast array of different solutions to be generated by multiple applications of this heuristic. For each route, r, in solution s_1 (the solution to be used first, s_1, is also determined randomly), a random number is generated between 0 and 1. If the random number for r is lower than x expressed as a decimal value, then r is included in the new solution. Following this, then an attempt is made to insert routes from the other parent solution into the child solution. The insertion is done sequentially, from the first route in the solution to the nth. Each route is inserted into the child solution if and only if it contains no customers that already exist in that child solution as a result of insertions from the first parent solution. Therefore all routes from the second solution that contain no conflicts with previously inserted routes are inserted into the child solution. Following this process, there will remain a set of customers who are not currently in the child solution; the next step of the heuristic is to insert these customers into the solution. These remaining customers are inserted in the order they appear in their original routes. The insertion position is determined through use of the insertCust method in Algorithm 5 (see the *shift* description). The resulting child solution is returned regardless of whether it improves upon the objective function scores of the parent solutions. See Algorithm 9 for pseudocode of this method.

Combine Long Combine Long is a crossover heuristic which shares the basic property of Combine in that it selects routes from both solutions before attempting insertion of remaining customers. The primary difference between the two is that, where Combine chose routes effectively at random for inclusion, Combine Longest selects routes based upon their perceived quality. In this case their quality is very simply defined by the number of customers served by that route. The logic behind this is that longer routes are preferable as they are more likely to lead to solutions with a lower overall distance and less routes needed also. This selection of high quality elements of a solution has the potential to yield strong child solutions. The first step of this heuristic is to create a set of all routes from both parent solutions and order this set by the 'size' of route. Following this ordering, the method works through the list of routes, in decreasing order, and attempts to insert the route into the child solution. As with Combine, a route will be inserted if and only if there exists no conflicts between that route and the routes already in the solution. A conflict, as above, is when the same customer exists in both routes. After selection of all possible routes, then any remaining un-inserted customers are inserted into the child solution in the same manner as for Combine. Hopefully it can be seen that Combine Long can be an effective heuristic by selecting elements of the parent solutions which are deemed to have strong qualities. Attempting to pick the longest routes from both solutions has the potential to create strong child solutions.

Algorithm 9 The *Combine* crossover heuristic which takes as input two solutions, s_1 and s_2.

```
procedure COMBINE($s_1, s_2$)
    Solution $first \leftarrow s_1$
    Solution $second \leftarrow s_2$
    Solution $newS \leftarrow createEmptySolution()$
    if $randomNumber() < 0.5$ then        ▷ Determine which solution will be considered first
        $first \leftarrow s_2$
        $second \leftarrow s_1$
    end if
    $randVal \leftarrow randomNumberBetween(0.25, 0.75)$
    for all Route $r$ in $first$ do
        if $randomNumber() < randVal$ then
            $newS \leftarrow newS : r$
        end if
    end for
    for all Route $r$ in $second$ do
        if $randomNumber() < randVal$ then
            if $noCustomerConflicts(r, newS)$ then
                $newS \leftarrow newS : r$
            end if
        end if
    end for
    List $UnroutedC \leftarrow determineUnroutedCustomers(newS, s_1)$
    for all Customer $c$ in $UnroutedC$ do
        $newS \leftarrow insertCust(c, newS)$
    end for
    return $newS$
end procedure
```

However, there is a danger that strong routes will share similar properties and customers and so conflicts could result in a large number of unrouted customers. As above, the resulting child solution is accepted irrespective of objective function value. The pseudocode for Combine Long can be seen in Algorithm 10.

Algorithm 10 The *Combine Long* crossover heuristic which takes as input two solutions, s_1 and s_2.

```
procedure COMBINE LONG($s_1, s_2$)
    Solution $newS \leftarrow createEmptySolution()$
    $Routes \leftarrow orderRoutesBySize(s_1, s_2)$
    for all Route $r$ in $Routes$ do
        if $noCustomerConflicts(r, newS)$ then
            $newS \leftarrow newS : r$
        end if
    end for
    List $UnroutedC \leftarrow determineUnroutedCustomers(newS, s_1)$
    for all Customer $c$ in $UnroutedC$ do
        $newS \leftarrow insertCust(c, newS)$
    end for
    return $newS$
end procedure
```

3.5 Conclusion

This chapter has addressed the question of what are the necessary components of a problem domain for hyper-heuristics. A basic definition of a problem domain has been given, which splits domain components into 2 categories, Problem Representation and Domain Tools. In addition, the question has been addressed of what constitutes a 'good' problem domain - in other words a problem domain that is suitable and powerful for hyper-heuristics to operate on. For this, the different components of a problem domain have been examined, with their relationships to the hyper-heuristics that use them being discussed and analysed. From this work, it has been established that there is a strong link between the tools provided by a problem domain and the workings of a hyper-heuristic or other adaptive algorithm.

To further demonstrate the factors that influence the design of a problem domain for hyper-heuristics, and to provide a base domain for further work within this thesis, a problem domain for the Vehicle Routing Problem with Time Windows has been presented. The motivation behind the decisions for the various domain components has been provided. The new contributions of the domain were described, including the combination of many of the most successful low-level heuristics from the literature, and the inclusion of 2 new crossover heuristics.

Chapter 4

Iterative Local Search Approaches to Cross-domain Optimisation

4.1 Introduction

The previous chapter began the main work of this thesis, i.e. the design of Vehicle Routing Problem domains to be used by hyper-heuristics. A definition of a problem domain was given, along with an analysis of the relationship between a domain's components and the hyper-heuristic that uses them. In order to allow informed designs of problem domains, it is also important to understand how the workings of a hyper-heuristic can influence how successfully it interacts with a domain. Hence, this chapter will address the question of 'Which qualities of a hyper-heuristic can impact on its ability to utilise the tools of a problem domain?'. In order to answer this, 2 hyper-heuristics will be proposed, with their origins being in differing research areas.

It is stated in the literature that an advantage to hyper-heuristics is their ability to adapt to different problems and instances, with minimal manual tuning needed. This poses an interesting question as to whether the qualities of a hyper-heuristic can have a different impact on performance when operating on multiple problems, as opposed to different instances of a single domain. Therefore, the work in this chapter will focus initially on the operation of hyper-heuristics when tested across 4 different problem domains. Following this, testing will be performed on a single problem domain, the VRPTW. Through these 2 approaches, analysis will be able to be performed on the ability of a hyper-heuristic to access a problem domain's tools successfully in both the context of multiple-domain and single-domain testing. The details of these methods, as well as the HyFlex domain described in the previous chapter, were accepted as a paper at the prestigious GECCO conference [27].

4.2 Previous HyFlex Work

The first mention of the HyFlex framework can be found in [20]. Part of the article is concerned with introducing the HyFlex framework and the initial four problem domains that have been described in the previous chapter, those of Personnel Scheduling, One-dimensional Bin Packing, Flow Shop and Maximum Satisfiability. Further to this, several simple hyper-heuristics are proposed and compared to an Iterated Local Search algorithm. In this case, the hyper-heuristics follow the definition given in the first chapter of a heuristic which selects other heuristics. All of the hyper-heuristics apply only the mutation heuristics as they are intended to represent iterative perturbation hyper-heuristics so other categories of low-level heuristic with more complicated operation are not required. For these hyper-heuristics, each iteration of their run requires two stages. Firstly, the selection and application of a low-level heuristic. From the definition of a hyper-heuristic given in Chapter 1, that of a heuristic to select heuristics, the method used to select low-level heuristics could be considered as the real 'hyper-heuristic' element of these algorithms. For the work in [20], two techniques were used, which will be described in detail below. Once the appropriate technique has chosen a low-level heuristic, it is applied to the solution. Following this is the second stage, solution acceptance. Solution acceptance is the task of determining whether a new solution obtained through the application of a low-level heuristic should be kept or discarded. Again, a number of methods were proposed, both deterministic and non-deterministic, which will be described below.

4.2.1 Heuristic Selection Mechanisms

- Simple Random (RN) This selection mechanism simply selects a low-level heuristic at random. There is no memory kept and no intelligence used. The authors of [20] used it as a benchmark method, but noted that it can be effective when paired with the correct solution acceptance criteria.

- Reinforcement Learning with Tabu Search (TS) This method, originally proposed in [29], includes two main elements. One is the tabu search implied by the name. Here, a tabu list is maintained which includes heuristics that cannot be considered for selection at that iteration of the algorithm. Low-level heuristics are placed into the tabu list if their application to a solution results in a deterioration of that solution's objective function value. The heuristic will only remain in the tabu list temporarily. In addition to the tabu search element of this algorithm is the reinforcement learning part. All the low-level heuristics are ranked, with the highest ranked heuristic being selected for application (assuming it is not in the tabu list.) As it is always the highest rank chosen, this method is deterministic and could also be described as greedy. Once a heuristic has been selected and applied to the solution, it will then have its rank either increased or decreased depending on whether it improved or deteriorated the objective function value for that solution. As mentioned previously, a reduction in objective function value will also result in the heuristic being placed in the tabu list.

4.2.2 Solution Acceptance Criteria

- Naïve Acceptance (NV) All improvements to the objective function result in the solution being accepted. Should there be a deterioration in value, the solution is accepted with a 50% probability. Again, this is a simple measure but one that has been effective on many problems.

- Adaptive Acceptance (AA) As with Naïve Acceptance, Adaptive Acceptance, which was proposed in [86] will accept all solution showing an improvement in objective function value. An element of memory and intelligence is used in determining whether deteriorations are accepted. This determination is controlled through an acceptance rate, which attempts to reflect whether or not the search is stuck in a local optimum. After every 0.1 second time period, the acceptance rate is either increased or decreased by 5%. It is increased if there is no improvement in objective function value, decreased otherwise.

- Great Deluge (GD) The Great Deluge method, proposed in [55], could be described as the most complex of the three acceptance criteria. It is analogous to a rising water level, with the water level broadly representing the acceptance rate. The level is determined by a formula, comprising elements including the current iteration of the search, the initial objective function value and the expected final value. The essence of this method is that initially almost all solutions are accepted; then, as the search progresses, fewer solutions are accepted until near the end of the algorithm, when only solutions with a value close to the final expected objective function value are accepted. This is a deterministic method.

4.2.3 Iterated Local Search

As the authors of [20] note, the term Iterated Local Search (ILS) was first proposed in [101]. The workings of the algorithm are simple to understand. It has three main elements to each iteration. First is the mutation stage which involves randomly selecting a low-level heuristic from a pool containing both the mutation and ruin-recreate categories of heuristics. This heuristic is then applied to a copy of the initial solution. Following this is the local search stage. For the work in [20] all of the local search heuristics are applied to the solution. The applications are performed independently and in a sequence that has been defined before the search. Once all local search heuristics have been applied to the solution copy, then there is the final solution acceptance stage. For this stage, the objective function value for the solution copy is compared to the value for the initial solution before mutation and local search heuristics were applied. If there has been an improvement in objective function value, then the new solution is kept. Otherwise the solution is discarded.

4.2.4 Experiments and Results

For the Burke et al article [20], three of the HyFlex problem domains were used for testing. These are Flow Shop, One-dimensional Bin Packing and Personnel Scheduling and for each problem domain 10 instances were chosen. The Iterated Local Search algorithm was tested against 6 hyper-heuristics. These 6 come from a combination of the 2 heuristic selection mechanisms and 3 solution acceptance criteria. For scoring, two methods were used. The first was to simply rank the algorithms for each instance on each problem domain. Therefore the best performing algorithm for one instance would have the rank 1. The second method used a score calculated by an equation used in the ROADEF operations research competition. The equation considers the best and worst scores on each instance when calculating a value. Full details can be found in [20]. Table 4.1 shows a summary of the results found from that work.

From the results, it is evident that even this simple version of Iterated Local Search is effective when compared with some hyper-heuristic approaches. Although it only performs best on one domain, its overall average is significantly better than the other algorithms'. It is also interesting to note that the adaptive solution acceptance mechanism works well for the hyper-heuristics. However, the randomised heuristic selection technique outperforms the tabu search approach, which could be considered as surprising.

Algorithm	Personnel Scheduling	Bin Packing	Flow Shop	Average
ILS	3.9 (76.03)	**1.7** (88.68)	2.7 (73.54)	**2.77 (79.42)**
TS+AA	3.5 (65.46)	2.2 (84.61)	4.8 (29.77)	3.50 (59.94)
TS+GD	6.0 (24.65)	2.7 **(92.43)**	2.9 (70.67)	3.87 (62.58)
TS+NV	3.4 (79.49)	5.0 (60.36)	4.9 (24.09)	4.43 (54.64)
RN+AA	2.5 (87.78)	3.5 (77.32)	3.9 (42.21)	3.30 (69.10)
RN+GD	5.6 (29.26)	6.6 (5.69)	**2.1 (87.66)**	4.77 (40.87)
RN+NV	**2.4 (90.63)**	6.3 (8.50)	4.9 (35.70)	4.53 (44.94)

Table 4.1: A table showing the results from the long track of the tests in [20]. The average rank over the 10 instances of each domain is given, as well as an average over all three domains. In brackets is the ROADEF score. Bold denotes best rank/score.

4.3 Adaptive Iterated Local Search

From the work performed in [20], outlined above, it would seem that Iterated Local Search (ILS) is a powerful algorithm with the potential to deliver robust performance across multiple problem domains. Given its superior results over the other hyper-heuristics presented, it would seem to be a good algorithmic framework from which to develop further. It would seem that there is scope to add greater sophistication to ILS. One such way of doing this would be to modify the way in which mutation heuristics are selected. In the paper above, the selection is made at random. However, the tools and information provided by the domain allow for information about the results of applications of a mutation heuristic to be used to intelligently select heuristics in future. Specifically, the tools that could be used are the objective function values returned from applications of heuristics. In addition the classification of heuristics into categories within HyFlex allows an algorithm to examine the results of sequences of operations. For example, the results of the application of a mutation heuristic, followed by a local search heuristic.

There is already precedent for adaptively modifying the mutation stage of ILS in the literature. In [163] an adaptive operator selection technique called Adaptive Pursuit [162] is successfully used to adaptively select the scale of perturbation to be made during the mutation stage of ILS. Although, that approach operated on the perturbation step size rather than selection of mutation heuristic, it shows the possibility of introducing adaptive measures into the mutation stage of ILS. In this chapter, two extensions will be proposed to the ILS presented in [20]. Both will concern how mutation heuristics are selected in the Iterated Local Search algorithm. The first will utilise a method named Extreme Value Based Adaptive Operator Selection [58], a method taht has been used in Evolutionary Computation algorithms, and the second is a hyper-heuristic named the Choice Function

[41]. The basic ILS algorithm used will be described in detail below followed by descriptions of both variants of mutation heuristic selection. The pseudocode for this ILS method can be seen in Algorithm 11.

4.3.1 Approach

Basic ILS Algorithm

The Iterated Search Algorithm used has the same basic framework as that described above. It iterates through the stages of mutation heuristic application, local search heuristic application and solution acceptance determination. The stages operate as below.

- Mutation Heuristic Selection There are two variants for this stage. See sections 4.3.1 and 4.3.1 for further details. For both variants, at each iteration a single mutation heuristic is selected and applied to a copy of the last solution. This modified copy is then passed to the local search stage of the algorithm. For this algorithm, a mutation heuristic is interpreted as a low-level heuristic from a pool containing both the mutation and ruin-recreate heuristics, as specified by HyFlex.

- Local Search Heuristic Selection In this local search stage of the algorithm, all of the local search heuristics are independently applied to the mutated solution from the previous stage. The resulting solution yielding the best (lowest) objective function score is kept, with the others being discarded. This method of application is known as *Best Improvement* although each individual local search heuristic still operates in a *First Improvement* manner.

- Solution Acceptance The solution acceptance criterion used for this algorithm is a simple greedy method. If the objective function value of the copy of the solution following the local search stage is superior (lower) than that of the original solution before mutation, then the new solution is kept. Otherwise it is simply discarded. The reason more sophisticated methods were not used is that introducing a number of acceptance criteria would have complicated analysis of results from testing as the main focus of the work is to analyse methods for selection of mutation heuristics. Also these more sophisticated methods have several control parameters that would have to be tuned, adding complexity to the algorithm and potentially compromising the robustness of the algorithm.

Algorithm 11 An adaptive iterative local search method for use in HyFlex.

```
procedure Adaptive Iterative Local Search
    Solution $s \leftarrow initialiseSolution()$
    $mutHScores \leftarrow initialiseScores()$
    while $timeHasNotExpired()$ do
        Solution $s' \leftarrow s$
        $mutHIndex \leftarrow selectMutationHeuristic(mutHScores)$
        $s' \leftarrow applyHeuristic(mutHIndex, s')$
        Solution $s'' \leftarrow s'$
        for $i \leftarrow 0, numberOfLSHeuristics()$ do
            Solution $s''' \leftarrow s'$
            $s''' \leftarrow applyHeuristic(lsHeuristic_i, s')$
            if $objFunc(s''') < objFunc(s'')$ then
                $s'' \leftarrow s'''$
            end if
        end for
        $s' \leftarrow s''$
        if $objFunc(s') < objFunc(s)$ then
            $s \leftarrow s'$
            $mutHScores \leftarrow updateScores()$
        else
            $mutHScores \leftarrow updateScores()$
        end if
    end while
end procedure
```

Extreme Value Based Adaptive Operator Selection

This first method for selecting mutation heuristic comes from the Evolutionary Computation literature and the area of *Adaptive Operator Selection (AOS) [58].* AOS has two main elements, *Credit Assignment* and the *Selection Mechanism. Credit Assignment* has its genesis in the 1980s [47] and is a term used to describe how an operator may be awarded or punished for their performance. In the context of this algorithm, performance is measured by the change in objective function value after the application of a mutation heuristic. Typically, the operator may be awarded or detracted points in proportion to the change in value. The method used here, *Extreme Value Based Adaptive Operator Selection (ExAOS)*[58], operates on the principle that it is preferable to achieve large, if infrequent, gains in solution quality rather than smaller yet more frequent gains. The method will now be fully explained. The process is shown in pseudocode in Algorithm 12. The credit assignment mechanism for ExAOS operates as follows.

For each operator (heuristic), a window of values is kept. Each value represents the change in objective function value from that application of the heuristic. The window is of size s representing the preceding s applications of the mutation heuristic. ExAOS assigns credit by selecting the largest value within the window of value (i.e. the greatest improvement in objective function value over the last s applications) and using this value as the credit to be given to that heuristic. The window size here can control the trade-off between the immediate performance of a heuristic and its past performance. Where the size is too low, important heuristics can be overlooked as their strong results are discounted too quickly. To the other extreme, a window size that is too large can cause single heuristics to dominate by virtue of having a single very strong iteration and can similarly lead to effective heuristics being overlooked. The value used for this implementation of ExAOS is 25, a value selected following preliminary testing of a number of possible values. This value has been obtained through testing of different values and is believed to provide a fair trade-off between the issues mentioned above. The choice of value could merit further investigation to establish the extent to which the performance of the algorithm depends on the value of the parameter.

The second element of the algorithm is the *Selection Mechanism.* At each iteration, this element selects an operator to be applied. The choice of operator is based upon their credit, as designated by the previous stage of the algorithm. The challenge for this element is to choose heuristics with a good chance of improving the solution, whilst occasionally allowing heuristics that have previously

Algorithm 12 The Extreme-value Adaptive Operator Selection method [58], which takes as input the current score windows for the mutation heuristics and returns the index of the heuristic to use.

```
procedure EXTREME VALUE ADAPTIVE OPERATOR SELECTION($mutHWindows$)
    $bestScore \leftarrow 0$
    $bestMutH \leftarrow -1$
    for $i \leftarrow 0, numOfMutationHeuristics()$ do
        if $bestScoreInWindow(mutHWindows_i) > bestScore$ then
            $bestScore \leftarrow bestScoreInWindow(mutHWindows_i)$
            $bestMutH \leftarrow i$
        end if
    end for
    $total \leftarrow 0$
    for $i \leftarrow 0, numOfMutationHeuristics()$ do
        if $i == bestMutHH$ then
            $total \leftarrow (total + (2 * bestScoreInWindow(mutWindows_i)))$
        else
            $total \leftarrow (total + bestScoreInWindow(mutWindows_i))$
        end if
    end for
    $mutHProbs \leftarrow 0$
    for $i \leftarrow 0, numOfMutationHeuristics()$ do
        if $i == bestMutHH$ then
            $mutHProbs_i \leftarrow ((2 * bestScoreInWindow(mutWindows_i))/total)$
        else
            $mutHProbs_i \leftarrow (bestScoreInWindow(mutWindows_i)/total)$
        end if
    end for
    $mutToUse \leftarrow rouletteSelection(mutHProbs)$
    return $mutToUse$
end procedure
```

performed poorly to be applied. The logic behind this is that some heuristics may perform poorly toward the start of the search but produce better results when the search is further on. Therefore, it is not desirable to punish a heuristic for early poor performance for the duration of the search. In practical terms, a *selection mechanism* will usually assign probabilities to the operators proportional to their credit. From these probabilities, an appropriate method will then be used to make the final selection of a heuristic. The specific method to be used has been chosen to fit the same philosophy as that behind the credit assignment mechanism. The *Adaptive Pursuit* selection mechanism, adapted to selection of operators in [162], assigns a higher probability to the operator which has the highest credit of all operators. The probabilities for the other operators are accordingly assigned as lower than their relative proportions would suggest. Again, this rewards the best of the best in terms of objective function improvement. Once these probabilities have been assigned, there is the question of how they should be used to actually select an operator. Three methods for this task were proposed in [41]. Of these the *Roulette Wheel* selection method is the one used here as it offers a simple but representative means of selecting operators.

For *Roulette Wheel Selection*, each of the operators is assigned a chunk of a figurative roulette wheel that is proportional to their probability value that has been given by *Adaptive Pursuit* compared to the other operators. A random number is then generated which is used to calculate at which point of the roulette wheel the metaphorical ball should stop at and hence, which heuristic should be selected for application. The pseudocode for roulette wheel selection can be seen in Algorithm 13.

Algorithm 13 The Roulette Wheel Selection mechanism which takes as input a set of probabilities. and returns an index representing the selection.

```
procedure ROULETTE WHEEL SELECTION($Probs$)
    $rand \leftarrow generateRandomNumber()$ ▷ Generate random number between 0 and 1.
    $total \leftarrow 0$
    for $i \leftarrow 0, sizeOfList(Probs)$ do
        $total \leftarrow (total + Probs_i)$
        if $rand < total$ then
            return $i$
        end if
    end for
end procedure
```

Choice Function

The second method for mutation heuristic selection is a fairly recent proposal from the hyper-heuristic literature. Named the *Choice Function* and first introduced in [41], this method considers several measures of performance when selecting a heuristic. They are described below.

- **f1** The first of these measures is the recent performance of a heuristic. This measure, *f1*, combines the changes in objective function value from the past n applications of the heuristic. A parameter α controls the balance between the most recent applications and those further back. The full formula can be found in [41]. It is not necessary to maintain a list of the result of previous applications as the information is automatically contained within the *f1* value.

- **f2** The second measure is *f2* and intends to capture how well pairs of heuristics operate together. That is to say, it may be beneficial to frequently apply heuristic x followed immediately by heuristic y as the changes they make to the search space work well together. In practical terms, this requires maintainance of a matrix of all combinations of heuristics. It is important to treat the application of heuristic x then y as different to the application of y then x. As with *f1*, a parameter is used to control the balance between recent performance and past performance. This parameter, β, is a floating point value between 0 and 1.

- **f3** The third and final measure is that of the time taken since the last application of a heuristic. For *f3*,time is represented as the number of nano seconds since the last application of the heuristic. This is to ensure that a heuristic doesn't get forgotten as it may improve later in the search.

As well as the individual parameters for *f1* and *f2*, there are parameters that balance the importance of the three selection measures. These parameters, α, β and ρ are floating point values that represent the importance of *f1*, *f2* and *f3* respectively within the final equation F that gives a score for each heuristic. The parameter values used in the original work of [41] are used again here as they have been selected carefully through experimentation and should represent a complementary set of values. These value are {0.9,0.1,1.5} for {α,β,ρ}. Other values were not tested at this point but could be a subject of future testing.

Once the method described above has generated a score for a heuristic, this score is then converted into a probability that represents the heuristic's score proportionally to the other heuristics. This is done in the same way as described for the AOS method above (see Algorithm 12). When probabilities have been generated for all operators, the roulette wheel selection mechanism described for *ExAOS* and in Algorithm 13 above can be utilised to select a heuristic to apply.

4.3.2 Experiments

Problem Domains and Instances

The four original HyFlex problem domains have been used for testing of these approaches. They are *Personnel Scheduling*, *One-dimensional Bin Packing*, *Permutation Flow Shop* and *Maximum Satisfiability.* The *Vehicle Routing Problem with Time Windows* domain proposed in the previous chapter is not tested as the domain was not available at this stage in the testing. For each of these domains, 5 instances have been selected. Tables 4.2, 4.3 and 4.4 show which instances have been used for 3 of the domains. For the *Flow Shop* domain, 5 of the more difficult instances were chosen to provide a strong base for testing.

Instance	Name	Staff	Shift Types	Length(Days)
1	BCV-1.8.2	8	5	28
2	BCV-3.46.1	46	3	26
3	BCV-A.12.2	12	5	31
4	ERRVH-B	51	8	48
5	MER-A	54	12	48

Table 4.2: A table showing the instances used for the *Personnel Scheduling* problem.

Instance	Name and Source	Capacity	No. Pieces
1	falkenauer/falk500-1[57]	150	500
2	falkenauer/bpt501-1[57]	100	501
3	schoenfield/schoenfieldhard1[8]	1000	160
4	1000/10-30/instance1[30]	150	1000
5	2000/10-50/instance1[30]	150	2000

Table 4.3: A table showing the instances used for the *One-dimensional Bin Packing* problem.

Test Details

As mentioned above, for each of the 4 domains, 5 instances have been used for testing, meaning that 20 instances in all have been used. The instances have been selected for each domain with reference to their components (e.g. number of machines) in order to get a varied range of problems. For each of these instances,

Instance	Name	No. Variables	No. Clauses
1	uf250-01	250	1065
2	sat05-486.reshuffled-07	700	3500
3	blocksworld/huge	459	7054
4	flat200-1	600	2237
5	s2w100-2	500	3100

Table 4.4: A table showing the instances used for the *Maximum Satisfiability* problem.

10 runs have been performed for each algorithm, with the runs having an individual length of 10 CPU minutes. This value was chosen to give sufficient time for the algorithms to efficiently manipulate the search space without being so long as to diminish the number of tests that can be run. The machine used for running these tests was a PC running the Windows XP operating system. The machine has a 2.33GHz Intel(R) Core(TM)2 Duo CPU and 2GB of RAM. Three algorithms were tested (two of which are variants of the same method). It is important to note that no tuning of any sort has taken place in between instances and domains. Therefore the algorithms are exactly the same versions for all tests. The algorithm variants are:

- *Uniform* This is the baseline version of the Iterated Local Search algorithm proposed in [20] and described in 4.2. The mutation heuristic selection for *Uniform* is made uniformly at random.
- *Adap-EV* This variant of the basic ILS algorithm uses the *Extreme Value Based Adaptive Operator Selection* method described in 4.3.1 and Algorithm 12 for the mutation stage of the algorithm.
- *Adap-CF* This varaint of the basic ILS algorithm uses the *Choice Function* method described in 4.3.1 for selecting mutation heuristics in that stage of the algorithm.

4.3.3 Results

Three forms of data analysis will be performed for the results. These are ordinal data analysis, distribution of best objective function values and proximity to best known solutions for the *Personnel Scheduling* domain.

Ordinal Data Analysis

One problem that can arise when comparing performance across multiple domains is that the scale of magnitude of objective function values can vary significantly for each domain. This can render many traditional comparison methods insufficient. Such a problem is discussed in [156] and a solution of scoring methods in an ordinal way is proposed. In a simplified form, ordinal scoring would result in algorithms being awarded scores based upon their rankings for each instance. An advantage of this system is that the magnitude of the objective function value is not used, only the performance of the algorithm relative to the other algorithms for the same instance is used. Therefore it is a scoring measure that can be accumulated across several domains and can be of particular use for situations such as the testing performed here. The actual method of scoring chosen for use here is the *Borda Count* method, proposed as a voting method by Jean-Charles de Borda in 1770. For each instance used for testing, the *Borda Count* method assigns each heuristic a score r_{ik} where $1 \leq r_{ik} \leq n$. In this equation, r_{ik} is the rank for algorithm k for instance i and n is the number of competing algorithms. To get a total score across all instances, the following formula is used;

$$\sum_{i=0}^{m} r_{ik}$$

where m is the number of instances. As this method uses the rank as a score, the winning algorithm will be the one with the lowest aggregate score. The lowest possible score across all 20 (m) instances from the 4 domains is 20; that is to say a rank of 1 on each instance. Breaking it down further, the lowest score per domain is 5 as there are 5 instances for each domain. To the other extreme, the maximum possible scores are 15 for a domain and 60 in total. Of the 10 runs for each instance, it is the average objective function score from these runs that is used to calculate an algorithm's rank for that instance. For analysis, the *Borda Count* scores for the individual domains will be considered and these can be seen in tables 4.5, 4.6, 4.7 and 4.8. Furthermore, table 4.9 and figure 4.1 show the total *Borda Count* scores for the algorithms over all domains.

Instance	*Adap-EV*	*Adap-CF*	*Uniform*
1	3	1	2
2	2	1	3
3	2	1	3
4	2	1	3
5	3	1	2
Total	12	**5**	13

Table 4.5: A table showing the *Borda Count* scores for the *Flow Shop* domain.

It can be noted from studying these results that the adaptive ILS variant that uses *Extreme Value Based Adaptive Operator Selection* to select mutation heuristics enjoys the overall strongest result. This algorithm has the lowest total

Instance	*Adap-EV*	*Adap-CF*	*Uniform*
1	1	2	3
2	2	3	1
3	1	2	3
4	1	3	2
5	1	2	3
Total	**6**	12	12

Table 4.6: A table showing the *Borda Count* scores for the *Bin Packing* domain.

Instance	*Adap-EV*	*Adap-CF*	*Uniform*
1	1	2	2
2	1	3	2
3	2	1	3
4	2	3	1
5	2	1	3
Total	**8**	10	11

Table 4.7: A table showing the *Borda Count* scores for the *Maximum Satisfiability* domain.

score over the four domain, with 15% less points than the *Adap-CF* algorithm and 24.4% less points than the original ILS algorithm, *Uniform.* Furthermore, The *Adap-EV* algorithm demonstrates best performance on 3 of the 4 domains, performing less well only on the *Flow Shop* domain. The Flow Shop instances are taken from a single source, whereas the other domains' instances are taken from multiple datasets. Therefore, the poorer performance on this one domain could be explained by the fact that there is less diversity in the instances and hence less opportunity for the supposed robustness of *Adap-EV* to thrive. However, the fact that the algorithm performs so strongly on the other domains suggests that it is better able to adapt to changing problems and search landscapes than the other tested algorithms.

It is now interesting to consider why *Adap-EV* outperforms the other approaches. It would seem clear that the addition of an adaptive mutation heuristic selection mechanism improves the algorithm over *Uniform* by reacting to the search in an on-line manner and intelligently using the available data to select appropriate heuristics. This demonstrates how a hyper-heuristic can successfully use the tools provided by a domain to navigate the search space of a problem. However, the *Adap-CF* algorithm also adaptively selects mutation heuristics yet still performs worse than *Adap-EV*. One difference between the 2 adaptive algorithms is that *Adap-CF* has more parameters to tune than *Adap-EV*. In fact, *Adap-EV* only has a single parameter, that of window size. For each parameter, it can be difficult to find a value that will work well over multiple problem domains. There-

Instance	*Adap-EV*	*Adap-CF*	*Uniform*
1	1	3	2
2	1	3	2
3	2	3	1
4	1	2	3
5	3	2	1
Total	**8**	13	9

Table 4.8: A table showing the *Borda Count* scores for the *Personnel Scheduling* domain.

Domain	*Adap-EV*	*Adap-CF*	*Uniform*
Flow Shop	12	**5**	13
Bin Packing	**6**	12	12
MAX-SAT	**8**	10	11
Pers. Scheduling	**8**	13	9
Total	**34**	40	45

Table 4.9: A table showing the total *Borda Count* scores across all domains.

fore, when there are more parameters, as with *Adap-CF*, algorithms can become over or under-tuned to particular domains. Further experimentation should be done to establish whether different parameter value would lead to a better performance for *Adap-CF*.

As was mentioned previously, the average result over 10 runs was used to calculate ranks. This measure was chosen in order to fit in with the goals of generality and robustness. In other words, it is desirable to measure performance as a whole and not just the odd good result. However, *Borda Count* results for the best of the 10 runs can be see in table 4.10. Note that the sum of the totals is different to before as there were some ties for instances. On the whole, it can be seen that the same ordering is maintained in terms of results and the magnitudes are roughly similar.

Domain	*Adap-EV*	*Adap-CF*	*Uniform*
Flow Shop	10	**5**	14
Bin Packing	**7**	11	10
MAX-SAT	**8**	9	9
Pers. Scheduling	**8**	11	10
Total	**33**	36	43

Table 4.10: A table showing the total *Borda Count* scores across all domains when using the best result over the 10 runs per instance.

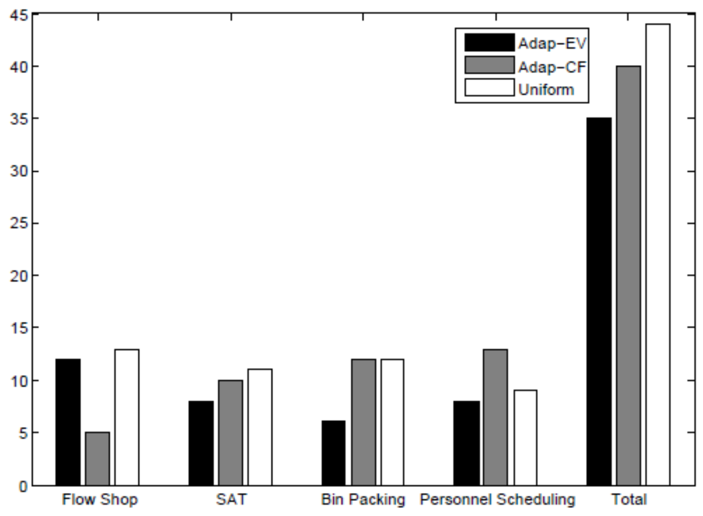


Figure 4.1: A graph showing the total *Borda Count* scores across all domains.

Analysis of Distribution of Objective Function Values

Not only is it important to analyse the performance of algorithms in terms of ranks and their numeric performance, but it can also be useful to visualise the whole range of results. For the testing performed here box plots have been used to view the distribution of objective function values for individual instances. One instance is shown for each problem domain with the plot for each algorithm demonstrating the distribution of values over the 10 runs performed for the instance. The 'box' part of the plot shows the upper and lower quartiles with a line indicating the median value. These plots can be seen in figures 4.2, 4.3, 4.4 and 4.5.

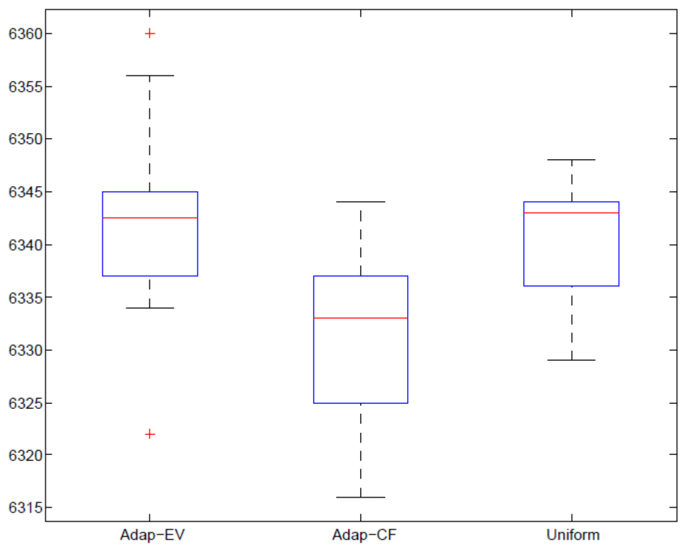


Figure 4.2: A plot showing the distribution of objective function values for instance 2 of the *Flow Shop* domain.

The same instance index (instance 2) was chosen for each domain so that the

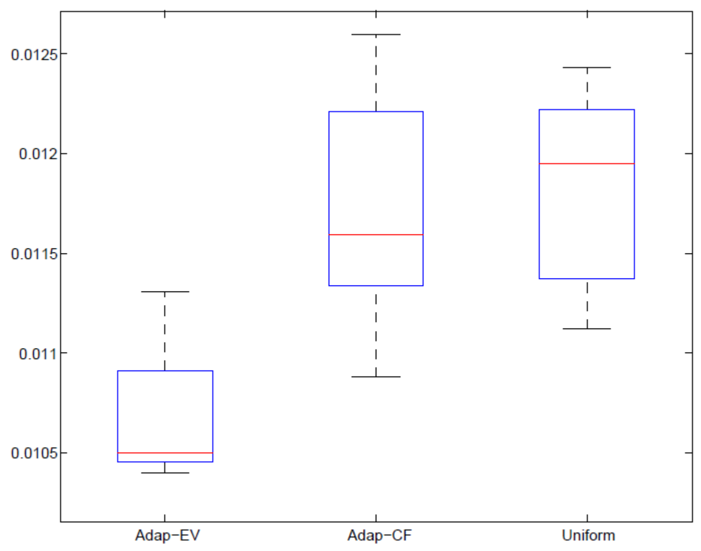


Figure 4.3: A plot showing the distribution of objective function values for instance 2 of the *Bin Packing* domain.

results have not been cherry-picked. It can be observed that the overall picture is similar to the results when analysed under the *Borda Count* method. It is once again the *Flow Shop* domain that provides the largest deviation from the perceived norm, with the *Adap-CF* method showing the strongest results. However, on the other domains, it is the *Adap-EV* algorithm that has the lowest median and lower quartile values. It is particularly interesting to see that in the *Bin Packing* domain, the *Adap-EV* algorithm has a very narrow box. This indicates that the results for this instance are within a narrow range which shows a high level of consistency from the algorithm; a quality that is desirable in terms of achieving robustness. Of course, the results have to be consistently strong, rather than just consistent and it is pleasing that, in this case, they are. The reason behind this could, again, be the lower number of parameters for *Adap-EV* which make the algorithm less sensitive to different initial solutions and to different solution landscapes.

Comparison Against Best Known Results for Personnel Scheduling Domain

So far, comparison of the algorithms has only been with respect to each other. Whilst useful, it doesn't necessarily indicate performance compared to other state-of-the-art work for the individual domains. It is worth bearing in mind that it is not entirely the goal of this work to generate new 'best-known' solutions on all problems tested, but more to generate solutions of high quality on a range of problems with little to zero manual tuning performed in between runs. However, it is still interesting to view how the performance does compare to the very best. A single domain has been selected for this comparison; the *Personnel Scheduling*

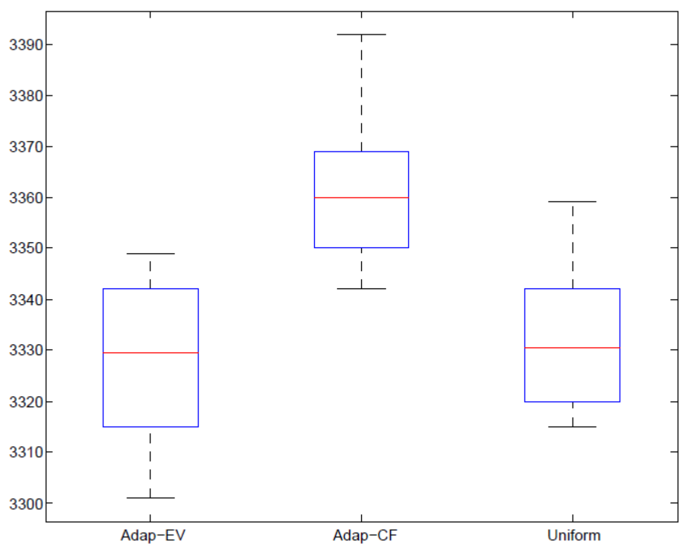


Figure 4.4: A plot showing the distribution of objective function values for instance 2 of the *Personnel Scheduling* domain.

domain. There are several reasons for choosing this domain. Firstly, that it includes a diverse set of complex instances including real world instances that provide a test for the algorithms. Also, the objective function values for the best known solutions are readily available (they can be found at [44]) meaning that comparisons can easily be made. The results for the best performing algorithm from above, *Adap-EV*, are compared to the best known results for the 5 *Personnel Scheduling* instances in table 4.11.

Instance	Name	*Adap-EV*	*Best-known*	% Deviation
1	BCV-1.8.2	**853**	**853**	0
2	BCV-3.46.1	3301	**3280**	0.64
3	BCV-A.12.2	2003	**1953**	2.56
4	ERRVH-B	**3177**	**3177**	0
5	MER-A	**9888**	9915	-0.27

Table 4.11: A table showing the best objective function values for the *Adap-EV* algorithm and the current best known results for the *Personnel Scheduling* domain.

The most interesting aspect of these results is that the *Adap-EV* algorithm has obtained a (at the time of presentation of [27]) new best known solution for one of the instances (*MER-A*). Being a general purpose algorithm which has not been specifically tuned for the *Personnel Scheduling* domain, it would not necessarily be expected to achieve best known solutions, so this is a promising result. It should be noted, however, that this result has since been surpassed and is no longer the best-known solution for this instance. As well as that one particularly strong result, there are two instance where *Adap-EV* was able to match the best-known solutions (*BCV-1.8.2* and *ERRVH-B*.) On the remaining two instances,

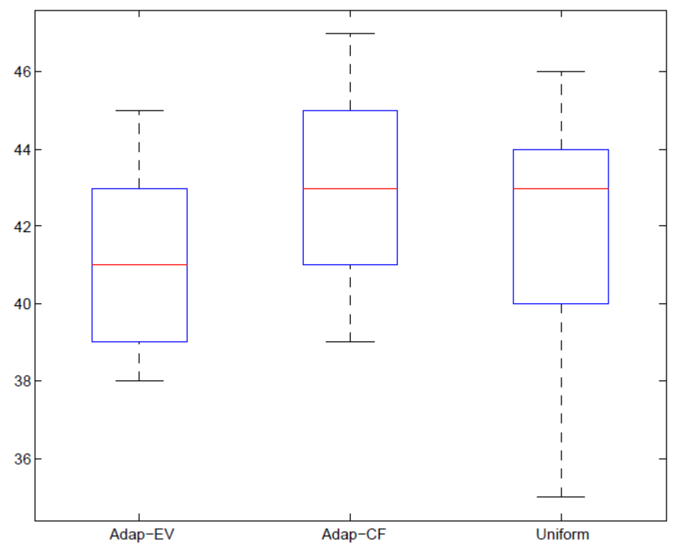


Figure 4.5: A plot showing the distribution of objective function values for instance 2 of the *Maximum Satisfiability* domain.

the percentage differences of the objective function values from the best knowns were low. Reading the results as a whole, there is significant encouragement that *Adap-EV* is an adaptable and robust algorithm. Credit must also be given to the problem domain for providing a strong set of heuristics with which to manipulate the search space.

4.4 Adaptive Iterated Local Search for the Vehicle Routing Problem

The work presented above showed how hyper-heuristics can utilise problem domain tools across multiple problem domains, by introducing an adaptive selection mechanism for mutation heuristics into an *Iterated Local Search* algorithm. The work to be presented now will pursue 2 natural extensions to this. Firstly, to devise a method for selecting local search heuristics that operates adaptively, as with the mutation heuristic selection. Secondly, to test these adaptive *Iterated Local Search* algorithms on the *Vehicle Routing Problem with Time Windows(VRPTW)* domain that was described in the previous chapter. This will enable analysis of the influence of hyper-heuristic design on performance on a single problem domain.

4.4.1 Ordered Local Search Improvement to AdapEV

The base algorithm for all 3 variants to be tested here is the basic *Iterated Local Search* algorithm described in section 4.3 above and named there as *Uniform*. One of the algorithms tested uses this exact algorithm, with no modifications. Another of them is the *Adap-EV* method described in the above section. The 3rd method applies a modification to the local search stage of Adap-EV and is described in this section.

Ordered Improvement Heuristic Selection within Iterated Local Search

In the previous section and for the first 2 algorithms to be tested here, the *Local Search* stage of the ILS algorithm operated in a *Best Improvement* manner. In short, all operators are independently applied with the resulting 'best' solution being kept. The modification to be made here uses an adaptive measure to select local search heuristics based upon past performance with the aim being that this use of search intelligence will result in improved solutions. Specifically, the *Local Search* stage for this method will now apply all operators to a solution in a particular sequence. This sequence is determined through reference to a heuristic's 'score'. As mentioned previously, this score is based upon a heuristic's past performance. To be precise, the measure used to calculate a heuristic's score is the mean improvement in objective function value yielded from that heuristic over all of that heuristic's previous application within the search. The heuristics are sorted in a decreasing order from the scores. Following creation of this sorted list, the *Local Search* heuristics are then sequentially applied to the solution. Each resulting new solution is kept and becomes the incumbent solution. The heuristics are applied sequentially until one of the applications fails to yield an improvement in objective function value or until all *Local Search* heuristics have been applied. The entire process of the local search stage can be seen in Algorithm 14 and is described below.

Algorithm 14 The ordered local search stage which takes as input a solution *s* and a set of scores, *LSScores*, for the local search heuristics.

```
procedure ORDERED LOCAL SEARCH($s, LSScores$)
    List $lsOrd \leftarrow orderValues(LSScores)$
    for $i \leftarrow 0, lsOrd$ do
        $startScore \leftarrow objFunc(s)$
        $s \leftarrow applyHeuristic(lsOrd_i, s)$
        $LSScores_i \leftarrow updateScore(startScore, i, LSScores_i)$
        if $objFunc(s) >= startScore$ then
            break
        end if
    end for
end procedure
```

- Sort all *Local Search* heuristics into an ordered list according to their mean improvement in objective function value.

- For all heuristics in the list, and in the order of the list, do the following:
  - Apply the heuristic to the incumbent solution, s_0.
  - Store the resulting new solution in s_0.
  - Update the heuristics score to include information from this previous application.
  - If there has been an improvement in objective function value, continue. Otherwise, end the *Local Search* stage.

4.4.2 Experiments

Problem Domain and Instances

The tests performed in the previous section used the 4 original HyFlex problem domains, *Flow Shop*, *One-dimensional Bin Packing*, *Personnel Scheduling* and *Maximum-Satisfiability.* As the *Iterated Local Search* algorithm was shown to be effective on these 4 domains, it will now be tested on the *Vehicle Routing Problem with Time Windows (VRPTW)* domain that was described in the previous chapter (chapter 3). As mentioned there, the VRPTW domain contains instances from 2 different sources, the Solomon dataset and the Gehring and Homberger dataset. The exact instances used are shown in table 4.12.

Instance	Name	No. Vehicles	Vehicle Cap.	No. Custs.
0	Solomon/RC/RC207	25	1000	100
1	Solomon/R/R101	25	200	100
2	Solomon/RC/RC103	25	200	100
3	Solomon/R/R201	25	1000	100
4	Solomon/R/R106	25	200	100
5	Homberger/C/C1-10-1	250	200	1000
6	Homberger/RC/RC2-10-1	250	1000	1000
7	Homberger/R/R1-10-1	250	200	1000
8	Homberger/C/C1-10-8	250	200	1000
9	Homberger/RC/RC1-10-5	250	200	1000

Table 4.12: A table showing the VRPTW instances to be used for testing.

Test Details

The three variants of the algorithms to be tested will be run on all 10 instances in table 4.12. There will be 20 runs for each instance and each run will last for 10 CPU minutes. These details mirror the settings used for the CHeSC competition and are designed to give an accurate representation of the quality of an algorithm. Three algorithms are to be tested and these are detailed below.

- *Rnd-ILS* This algorithm is exactly the same as the *Uniform* algorithm from the previous section. It is the Basic ILS method with no adaptive selection used in either the *Mutation* or *Local Search* stages.
- *Ad-ILS* This is the same algorithm as *Adap-EV* in the previous section. It uses an adaptive operator selection method to select mutation heuristics in that stage of ILS.
- *AdOr-ILS* This algorithm is the same as *Adap-EV* for all elements except the *Local Search* stage. For that stage, the method described above is used whereby each heuristic is assigned a score and ordered by this score. The heuristics are then applied in this order.

4.4.3 Results

The results will be analysed in three ways in order to give a complete picture of the algorithms' performance. These are described separately below. It should be noted that these methods haven't been compared to some of the more advanced in the literature (i.e. Misir et al. [110]) as many of those methods were proposed after this work was completed.

Analysis of Mean Objective Function Values

The first form of analysis involves examining the actual objective function values obtained from the runs. Table 4.13 shows the average objective function value over all 20 runs for each instance along with the standard deviation.

Instance	*AdOr-ILS*	*Ad-ILS*	*Rnd-ILS*
0	$\mathbf{5281.71}_{334.614}$	$5406.48_{404.159}$	$5292.43_{337.186}$
1	$21291.89_{482.56}$	$21212.60_{509.28}$	$\mathbf{21054.87}_{500.73}$
2	$\mathbf{13605.03}_{451.64}$	$13932.67_{616.29}$	$13827.54_{516.39}$
3	$\mathbf{6564.42}_{554.77}$	$7055.26_{748.15}$	$6760.62_{597.41}$
4	$\mathbf{14280.79}_{319.54}$	$14549.22_{449.1}$	$14600.09_{471.7}$
5	$\mathbf{155305.46}_{6154.24}$	$163041.76_{11226.39}$	$180301.07_{2921.14}$
6	$\mathbf{77302.72}_{3384.83}$	$79175.63_{3431.57}$	$82316.66_{2326.49}$
7	$\mathbf{163177.74}_{2100.09}$	$164341.16_{1550.06}$	$169729.31_{1721.3}$
8	$\mathbf{158941.93}_{2460.71}$	$163332.72_{4314.93}$	$172007.42_{2055.46}$
9	$\mathbf{149447.68}_{1500.9}$	$150276.89_{1644.28}$	$153648.66_{1079.4}$

Table 4.13: A table showing the average and standard deviation values for 20 runs each of 10 VRPTW instances. Best result for each instance shown in bold.

From these results, it is clear to see that the *AdOr-ILS* algorithm with adaptive ordering of the *Local Search* heuristics returns the best average objective function values for 9 out of 10 VRPTW instances tested. The only instance where it is outperformed is one of the simpler Solomon instances. On all of the more difficult Gehring and Homberger instances, it is the fully adaptive *AdOr-ILS* that performs best. From these results, it can also be observed that the same pattern is detected as with the other domains. That is to say, the *Ad-ILS* algorithm with adaptive selection of mutation heuristics outperforms the basic ILS algorithm, especially for the latter 5 more difficult instances. Furthermore, by examining the standard deviations obtained for all algorithms, it can be seen that the averages for the best performing *AdOr-ILS* method are not reliant on single very good values but in fact reflects a consistently strong performance. As well as producing lower averages, the *AdOrILS* algorithm also enjoys a strong performance in terms of the 'best' results over the 20 runs, as is shown in table 4.14. As can be observed, the *AdOr-ILS* algorithm in fact obtains the best 'best' result in all 10 of the instances tested. It is also noticeable that for the harder Gehring and Homberger instances, the gaps between the adaptive ILS variants and the original basic version are vast.

Instance	*AdOr-ILS*	*Ad-ILS*	*Rnd-ILS*
0	**5090.66**	5096.18	5100.01
1	**20637.71**	20643.08	20651.79
2	**13298.17**	13353.00	13365.45
3	**5316.05**	6242.36	6231.25
4	**13319.50**	14262.90	14266.72
5	**145639.37**	148627.86	174262.54
6	**71574.48**	71893.52	77418.49
7	**160007.17**	161407.06	165921.55
8	**154676.61**	156902.59	169691.21
9	**146929.46**	147607.07	151656.56

Table 4.14: A table showing the best values obtained from 20 runs each of 10 VRPTW instances. Best result for each instance shown in bold.

Box Plot Analysis

As with the results for the previous work, a graphical form of analysis will be presented here in the form of box-plots. These will allow examination of all 20 runs of an instance as a whole, rather than just viewing the average or best results. 4 instances have been selected for representation in a box-plot. All of these instances are from the harder Gehring and Homberger dataset. As before, the box plot includes line representing the lower quartile, median and upper quartile values, as well as crosses for outliers. The plots can be seen in figures 4.6, 4.7, 4.8 and 4.9.

One thing shown by these plots is the magnitude of the difference between results for the adaptive ILS algorithms and for the basic ILS method. Furthermore, if the 'heights' of the boxes are examined, it can be seen that the range of results for

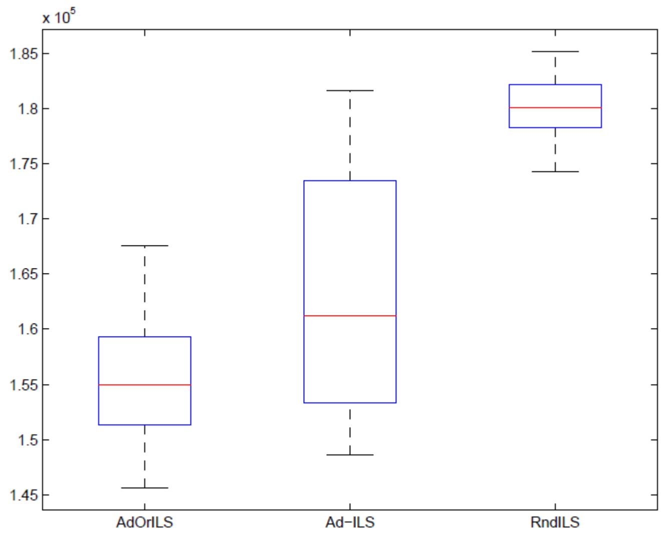


Figure 4.6: A plot showing the distribution of objective function values for instance 5 of the VRPTW domain.

the *AdOrILS* algorithm are within a tighter band than for the *Ad-ILS* algorithm so it would seem that the addition of an adaptive element for the *Local Search* stage has yielded a greater consistency of result as well as a better overall quality.

Statistical Significance Tests

In addition to stating that the improvements made to the *Local Search* stage of ILS have resulted in improved results in strict numeric terms, it is useful to declare whether or not these results are at a level that can be statistically said to be superior to the previous methods. Specifically, it is of interest as to whether the newly described *AdOr-ILS* algorithm is statistically better than the *Ad-ILS* algorithm. To test this, the two-sided *Wilcoxon Signed Rank* test will be used. The *Wilcoxon Signed Rank* test is a paired difference test that considers differences between ranks for pairs of data. In this case, the pairs of data are individual runs for the 2 algorithms to be examined. The null hypothesis for this test is that there is no difference between the algorithms. The test will be performed at a 95% confidence level, meaning that a p value of less than 0.05 results in a rejection of the null hypothesis. Therefore any instance that returns a p value of less than 0.05 can be said to represent an instance for which the *AdOr-ILS* algorithm is statistically superior to *Ad-ILS*. Table 4.15 ahows the p values for the 10 VRP instances tested.

From this table, it can be seen that statistical significance has been achieved in 7 out of the 10 instances tested. This would imply that there is a strong case for saying that the updated ILS with the addition of an adaptive *Local Search* selection mechanism (*AdOr-ILS*) is superior to the previous version (*Ad-ILS*).

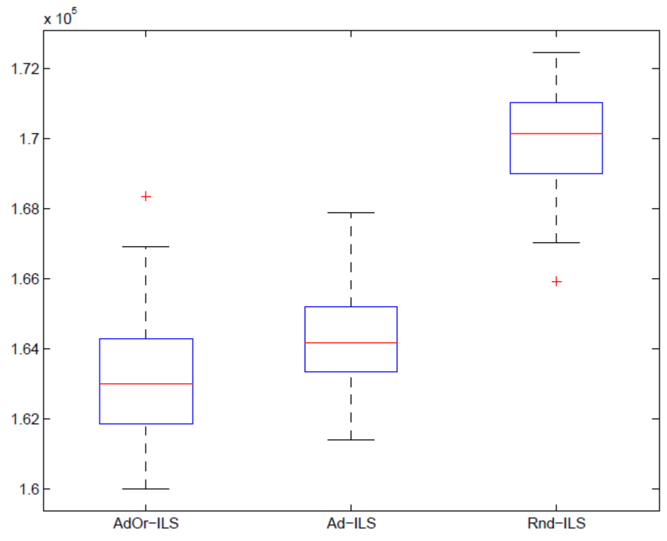


Figure 4.7: A plot showing the distribution of objective function values for instance 7 of the VRPTW domain.

Instance	0	1	2	3	4	5	6	7	8	9
p-value	**0.017**	0.455	**0.023**	**0.021**	**0.005**	**0.04**	0.086	**0.048**	**0.005**	0.126

Table 4.15: A table showing results from the *two-sided Wilcoxon Signed Rank Test* for the 10 VRP instances. A *p-value* of less than 0.05 indicates a statistical difference. This is highlighted in bold.

4.5 Conclusion

This chapter turned the tables on the question of problem domain design by considering how the design of a hyper-heuristic can influence its ability to successfully utilise the tools provided by a problem domain. This question was sub-divided to consider design for both multiple-domain and single-domain applications. Two hyper-heuristics were presented, which demonstrated in different ways how some domain tools could be used by a hyper-heuristic. The main domain tools utilised here were the classification and access to low-level heuristics as well as the information received from the objective functions of domains. From the testing performed on the 4 original HyFlex domains, it was shown that an *Iterative Local Search* approach with an adaptive mutation heuristic selection mechanism based on an *Adaptive Operator Selection* method performed better than one which used a *Choice Function* method. Both these methods attempted to use information gained from the search in order to make intelligent heuristic selection decisions. The largest difference between these methods could be observed in the number of parameters used for each algorithm. The extra parameters required for the *Choice Function* method seemed to make it more difficult for the algorithm to adapt to different problem domains. This would seem an interesting contribution - that the number of a parameters used by a hyper-heuristic can impact on its

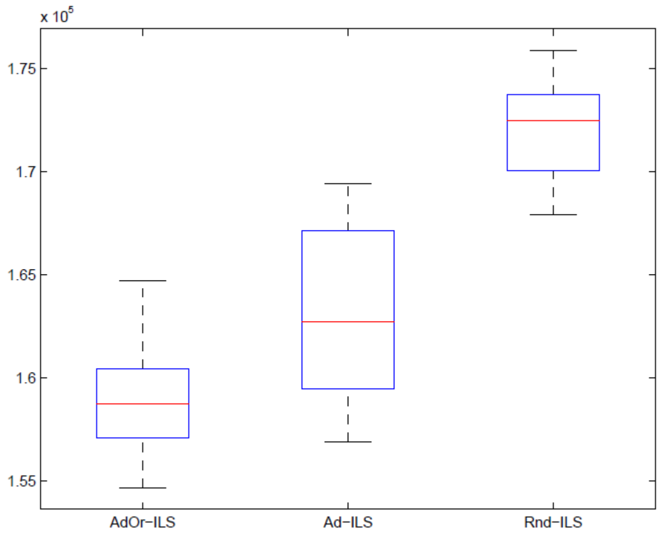


Figure 4.8: A plot showing the distribution of objective function values for instance 8 of the VRPTW domain.

ability to adapt to different problems when performing cross-domain optimisation.

As well as this cross-domain analysis, testing was performed on the *VRPTW* domain presented in the previous chapter. For this, the *Iterative Local Search* algorithm used for the cross-domain testing was augmented with an adaptive local search heuristic selection mechanism. The question was whether using this extra 'information' would allow the hyper-heuristic to better navigate the problem search space. Results using this new approach demonstrated that the algorithm was able to do this successfully and improve upon the results obtained by the *Iterative Local Search* method which didn't have the adaptive local search stage. These results further show the relationship between a problem domain and a hyper-heuristic. A problem domain must offer the tools, or information, to allow modification and imporvement of solutions. However, the hyper-heuristic must still be designed in such a way as to take advantage of these tools. The next chapter will investigate further how this offering of extra information by a problem domain can benefit a hyper-heuristic.

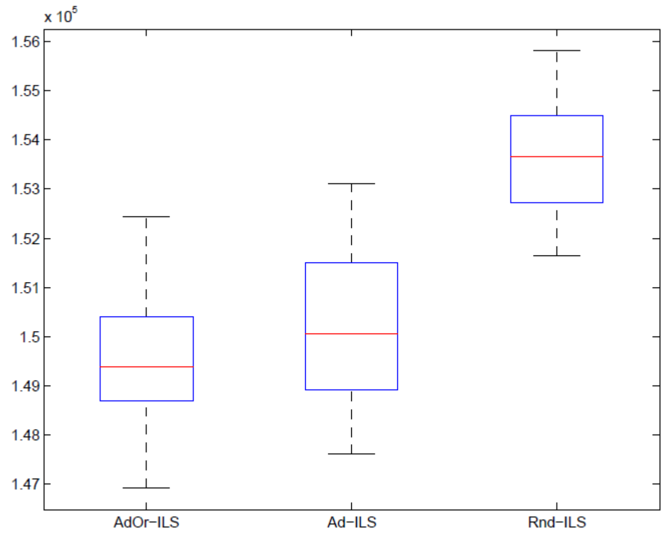


Figure 4.9: A plot showing the distribution of objective function values for instance 9 of the VRPTW domain.

Chapter 5

An Adaptive Memetic Algorithm and Extensions to the Hyflex Framework

5.1 Introduction

The previous chapter showed how tools and information offered by a problem domain to a hyper-heuristic could be used to navigate the search space and improve solutions. This chapter will build further on this work by addressing the question 'How can extra tools and information offered by a problem domain be used by a hyper-heuristic to improve results?'. To this end, a number of new additions will be proposed for the HyFlex framework, designed to provide more data to the hyper-heuristic to allow it to make more informed search decisions. One of these additions in particular, a measure of 'distance' between two solutions, will be looked at in detail. By using this feature that is present in some genetic algorithms, we will be able to analyse whether it can be provided in such a way as to be useful to a hyper-heuristic also. These extensions and the algorithmic approach described in the chapter were presented at the PPSN conference [117].

So far, the work that has been presented in this thesis has concentrated on single-point hyper-heuristics. It is important to also consider how domain design impacts upon performance of population-based approaches. There is a key question about whether population-based methods are more, less or equally able to use the tools provided by a domain during the search. To achieve this, a population-based hyper-heuristic will be proposed and compared to the *Iterative Local Search* of the previous chapter. Analysis will be performed of their interactions with the domain.

5.2 Improvements to HyFlex framework

As was mentioned in chapter 3, one of the main motivations behind the introduction of HyFlex was to provide a means for designers of hyper-heuristics to test their algorithms across multiple problem domains and over a wide set of instances. From work previously done that uses HyFlex and from the entrants of the first CHeSC competition that uses the HyFlex framework (see section 5.6), it can be seen that the term *hyper-heuristic* can be used to describe a vast range of meta-heuristic approaches, from single-point multi-stage algorithms such as *Iterated Local Search* to population-based approaches such as *Evolutionary Algorithms.* From the definition of a hyper-heuristic given previously, *'a heuristic which selects other heuristics'*, it can be stated that any algorithm that has an element of heuristic selection (even an extremely simple method such as random selection) can be considered a hyper-heuristic. The challenge HyFlex has is to ensure that the correct range of tools are available for all varieties of algorithm. To this end, a number of improvements and additions to the HyFlex framework are proposed (originally described in [117]) with the intention of permitting a greater range of methods to be implemented within the framework and a greater amount of data to be available to the hyper-heuristics. Below, the modifications will be individually described with their motivations and practical workings. Following that will be a full explanation of how the changes have been made for the *Vehicle Routing Problem with Time Windows (VRPTW)* domain.

5.2.1 Additions

Distance between Solutions (Solution Diversity)

One branch of algorithms that can be well suited to an implementation in HyFlex are *Evolutionary Algorithms.* As well as allowing a population of a size determined by the user, the separation of different types of heuristics in HyFlex facilitates development of this type of algorithm where, typically, different stages of the *Evolutionary Algorithm* will require a different category of heuristic - for example, mutation or crossover. However, there is one piece of information commonly used in such algorithms that has been previously inaccessible in HyFlex, that being a measure of the genotypic diversity between two solutions. Crucially, this can be an important piece of information when assessing performance of mutation heuristics. As its name would suggest, the purpose of a mutation heuristic is to modify the solution in some way. It is not possible to deduce to what extent it has done this without some measure of how 'different' a solution is to before the modification. For example, in HyFlex, the *intensityOfMutation* parameter is used to determine to what extent a mutation heuristic will modify a solution. Using a distance measure, the relationship can be studied between the parameter value and the mutation levels achieved by the heuristic. If increasing the parameter value from 0.1 to 1 only provides a small increase in the extent to which a solution is mutated, it could be implied that the affect of the parameter is small and its value could be adjusted appropriately. This information is provided through HyFlex by the addition of 2 methods to the interface, as described below.

- **double getMaxDistance()** This method is provided to account for the fact that various distance measures will return results of different scales, not neccessarily between 0 and 1. By providing a max value, the hyper-heuristic designer can accurately analyse the distance value returned. The method returns the maximum distance value, *maxdist* that can be returned by the **distanceSolutions** method. It is provided to give greater information to users for domains where distance may not be measured purely between 0 and 1.
- **double distanceSolutions(int solutionIndex1, int solutionIndex2)** Returns a value between 0 and *maxdist* that represents the genotypic distance between solutions *solutionIndex1* and *solutionIndex2*. A value of 0 indicates identical solutions and a value of *maxdist* represents the maximum possible distance between solutions as measured by this method. A full description of its implementation for this domain will be provided later in the chapter.

This is the first time a distance measure has been added to HyFlex and, as such, there is scope for the concept to be extended in the future. One way to do this would be to allow multiple distance measures to be available for each domain. The hyper-heuristic could then select which measure to use; possibly in an adaptive measure akin to those used for heuristic selection. As discussed in the introduction to this chapter, the key to the success of a hyper-heuristic lies in the data available to it and how it uses this data. By increasing the amount of data available to a hyper-heuristic, there is potential for an increase in performance.

Access to Features of Objective Function

Previously in HyFlex, only a single value has been used to judge performance, that of the objective function. For many problem domains and instances it may be the case that this objective function has been composed of several measures or scores for different 'features' of a solution. The objective function will often be a weighted sum or assessed as a hierarchy. A solution feature can be defined here as a non-trivial property of a solution. However, there are several reasons why the designer of a hyper-heuristic or meta-heuristic may wish to know the values for individual features. Firstly, should they wish to check the performance of their algorithm against best known solutions, it is often necessary to break the objective function down into its various components. As an example, solutions for the *Vehicle Routing Problem with Time Windows(VRPTW)* are judged for quality primarily by examining the number of routes; then by considering the total distance travelled. For hierarchical objectives such as this, allowing these feature values to be easily accessed will faciliate assessment of an algorithm's quality.

A second reason for wishing to access solution features may be in order to devise different objective functions to the standard one used within a HyFlex domain. For some problems, it may be the case that different objective functions are used for various instances or variations of a single problem. Further, the objective function can often be used as an important driving force within the search. In [5], the authors explore how modifications to objective functions and weights within objective functions can be used to escape local minima within several problem domains, including approaches in the *Travelling Salesman Problem* [168] and the *Maximum-Satisfiability Problem* [146][91]. Some examples have used individual solution features in isolation as the objective function for different parts of the search. As an example, in [9], the first half of the search uses a *Simulated Annealing* approach to minimise the number of routes for *Pickup and Delivery Vehicle Routing Problems with Time Windows and Multiple Vehicles.* The second half of the search uses a *Large Neighbourhood Search* to minimise the total distance travelled.

Finally, the work of Knowles et al. in [93] demonstrates that, even in a single-objective problem, local optima can be escaped by transforming a problem into a multi-objective problem. These examples show how important a new feature this is and that it is a valuable new addition to the framework.

Two new methods have been added to the HyFlex interface to include these new features. Again, their exact implementations for the VRPTW domain will be described in a later section. The methods are:

- **int getNumberOfFeatures()** This method identifies the number of solution features available to the hyper-heuristic designer within the problem domain. It returns an integer which represents the number of features for this particular problem and instance.

- **double getFeatureCost(int solutionIndex, int featureIndex)** Returns the value for the specified feature within the specified solution. The problem domain documentation should state which feature has which index.

External Instances

While individual HyFlex domains aim to provide a wide set of problem instances from a variety of sources, it is inevitable that they will not contain all instances from all datasets. Furthermore, it can often be the case that an algorithm designer will create their own variants of instances in order to test their method on particular search landscapes. Therefore it would be desirable for HyFlex to include a function which allows a user to upload their own instance for use within the framework. For this purpose, a single method will be added (see below) that allows a user to do this. The instance must follow the same file format as the other instances for the problem domain. The following method is the one to be used for this new feature:

- **boolean loadInstanceFromFile(String fileName)** Loads an instance from the location *fileName* in the local file system in the same manner as an instance is loaded by index. This instance becomes the current instance to be operated on. True is returned if the instance is successfully loaded. False is returned if the loading has not been successful.

Saving and Loading of Solutions to/from Files

There are several reasons why an algorithm designer may wish to save a solution to a file and/or load it into a search at a future date. One example may be the case where one algorithm is originally used to operate on a solution, before a second algorithm is then utilised. An additional example would be where multiple single point searches are used to generate a number of solutions to be saved. These solutions could then be loaded into an Evolutionary or Genetic algorithm as a seed population. Through methods such as these, this can prove an important addition to HyFlex which can improve solution quality. Two methods have been added to the framework to allow this. They are as follows.

- **boolean saveSolutionToFile(String fileName, int solutionIndex)** This method will save the solution currently at index *solutionIndex* within the HyFlex domain into a file to be saved at location *fileName* within local hardware memory. True will be returned if the operation is successful and false otherwise.

- **loadSolutionFromFile(String fileName, int solutionIndex)** This method will load a solution from the local file *fileName* into the domain at index *solutionIndex*. True will be returned if the operation has been successful and false otherwise.

5.2.2 Definition of 'Distance' for HyFlex Domains

The following section of this chapter will describe an implementation of the HyFlex additions for the VRPTW domain. Firstly, though, this section shall contain a discussion of how distance could be defined for the initial 4 HyFlex domains - *Permutation Flow Shop*, *Personnel Scheduling*, *One-dimensional Bin Packing* and *Maximum Satisfiability (MAX-SAT)*. By providing definitions for all domains, the contribution to understanding can be extended and future implementation can be facilitated. These will now be discussed separately below.

Permutation Flow Shop

Permutation Flow Shop is the problem of ordering a set of n jobs which have to be processed on a set of m machines in a set sequence. Two distance measures for this problem are proposed by Portmann and Vignier in [128]. One is base on common edges between solutions. As this problemm is a permutation-based representation, an edge can be considered as two consecutive jobs within a solution. If both solutions have these consecutive jobs in the same order, it is considered a common edge. The relative position of the jobs is not relevant. The second method considers precedence of jobs and looks for situations where similar precedence of jobs is maintained between solutions. Both of these measures proposed in [128] are designed to work on a 2 parent, 1 child situation. However, for distance in HyFlex we are only comparing two solutions. The second measure described above has different points awarded depending on whether the child solution has matching precedence to one, or both of the parents. This points system would be watered down in the HyFlex version and potentially be less effective. What we propose for this domain is to use the first method described above to compare common edges between solutions as this would not be adversly impacted by only having 2 solutions to compare. The following formula would be used to calculate a score between 0 and 1 for distance:

$$distance = \frac{totalEdges - commonEdges}{totalEdges}$$

Personnel Scheduling

The *Personnel Scheduling* domain in HyFlex is unique in that it allows for representation of many different permutations of nurse rostering problems, with many different constraints and objectives being permitted. As such, it is difficult to use standard literature methods in this context. A new distance measure for the HyFlex domain is proposed here. This measure compares the number of common shifts between 2 solutions. To explain further, a common shift would be a case where, in both solutions, the same employee has a shift of the same length at the same time. The formula below (which is similar to that for the Flow Shop domain) gives a value between 0 and 1:

$distance = \frac{totalShifts - commonShifts}{totalShifts}$

One-dimensional Bin Packing

The *One-dimensional Bin Packing* problem is the problem of fitting a number of items with different weights into bins of a fixed size. It would seem that, as the ordering of items within a bin is not important, it would not make sense to use an edge-based approach as above. Instead, it is proposed to consider the weights of each item within a bin. If a bin in one solution has a set of items with weights $w_0..w_\mathrm{k}$ where k is the number of items in the bin, then there is a 'common bin' if there is a bin in the other solution with an identical set of items and weights. This will be quick to compute and give a simple measure of similarity to the hyper-heuristic. Again, the following formula is used to give a score between 0 and 1:

$distance = \frac{totalBins - commonBins}{totalBins}$

Maximum Satisfiability (MAX-SAT)

The MAX-SAT problem is that of determining a set of values for boolean variables to attempt to satisfy the maximum number of boolean clauses. Owing to its simple representation of boolean string, an ideal distance measure for this domain may be to follow the approach of Zhang et al. in [173]. There, the Hamming distance is suggested. The Hamming distance calculates the number of identical bits in two strings (taking into account position also). This would be useful for MAX-SAT as the Hamming distance will count the number of variables with the same boolean variable in bothe solutions. As above, the below formula can be used to translate the Hamming distance into a value between 0 and 1:

$distance = \frac{totalVariables - identicalVariables}{totalVariables}$

5.2.3 Implementation of Additions in VRPTW Domain

The previous section detailed the additions that have been made to the HyFlex framework. This section shall explain how these additions have been implemented for the *Vehicle Routing Problem with Time Windows (VRPTW)* domain. The original domain is described in chapter 3.

Distance between Solutions (Solution Diversity)

The aim of the implementation of a distance measure for the VRPTW domain is to provide an accurate measure of solution distance without the need for lengthy computation. In [95], several methods are proposed and analysed for measuring the genotypic distance between solutions for the *Capacitated Vehicle Routing Problem (CVRP)*. Among the proposed methods is one that was originally proposed in the *Travelling Salesman Problem (TSP)* literature [14]. For the TSP, this measure evaluates the number of common edges between 2 solutions in order to give a 'distance' value. An edge in the TSP can be defined as a link between two cities/nodes. This is the same for the CVRP and for the VRPTW with the slight clarification that it is an undirected link between 2 customers (or possibly a depot). A slight simplification of the method proposed in [95] is used here, in order to allow for quick execution during run-time. The formula to calculate the 'distance' value is as follows.

$$distance = \frac{totalEdges - commonEdges}{totalEdges}$$

This formula gives a value between 0 and 1 (hence the value returned for *maxdist* by the *getMaxDist()* method is 1) for the genotypic distance between two solutions.

Access to Features of Objective Function

There are three solution features available for this VRPTW domain. They can be accessed through the *getFeatureCost()* method with the appropriate index. The features provided are designed to give an algorithm sufficient data to be able to control the search and objective function to a desirable level. In addition, the features provided allow simple comparison to best-known solutions for the VRPTW instances provided and any external instances that compare performance using the same measures. The features are as follows.

- *No. of Routes (Index 0)* This feature represents the number of routes (or vehicles) used in the current solution. It is included as it is the primary objective to be minimised for the majority of benchmark instances for the VRPTW.

- *Total Distance Travelled (Index 1)* This feature is included as the distance is often used as the secondary objective for the VRPTW. This feature represents the total distance travelled over all routes, including travel from and to the depot. As a reminder, the distance between two customers (or a customer and a depot) within the context of this domain is defined as the euclidean distance between the co-ordinates of the customers (or depot).
- *Length of Shortest Route (Index 2)* This feature represents the minimum length of a route in the current solution. Length here is defined as the number of customers in a route plus the number of visits to the depot (always 2). This feature is included as it can be a useful measure in driving the search through the objective function. By identifying whether a route has a very low 'length', it can be decided whether the search is close to removing a route.

External Instances

Any VRPTW instance following the Solomon file format can be loaded into this domain. The format is explained in detail in section 3.4.2 of Chapter 3 but a brief summary is given below.

- **Name of Instance**
- **Vehicle Constraints** Number of vehicles, vehicle capacity.
- **Customer Data** Customer number, X co-ordinate, Y co-ordinate, demand, ready time, due date, service time.

Saving and Loading of Solutions

In order to save and load solutions to/from files, the methods described above (*saveSolutionToFile()*, *loadSolutionFromFile*) should be used. The solution will be saved in a format that, for each customer in each route, stores information such as the arrival time and waiting time for that customer as well as the general information such as location id and demand.

5.3 Population-based Approach to the VRPTW which uses Solution Distance

As discussed in the introduction, there will be two main goals for this chapter. The first will be to show that a population-based algorithm can operate effectively on the HyFlex framework in general and, in particular, on the VRPTW domain. This will be judged in reference to its performance when compared to the best *Iterated Local Search (ILS)* approach described in the previous chapter. Further comparisons will be drawn between the new approaches and current best-known data for the instances used. In addition to these two comparisons, the results from the algorithms will be pitched against the best-performing competitors from the first CHeSC hyper-heuristic competition to illustrate whether strong performance has been obtained against a wide variety of high-quality algorithms. The second goal is to demonstrate a way in which the additions to the HyFlex framework may be used to improve performance and to implement a wider range of algorithms. Specifically, an algorithm variant will be designed that utilises the new measure of solution distance in order to assess the effectiveness of the mutation heuristics that have been applied. This is an approach that would not previously have been possible to implement in HyFlex.

In this section, the algorithms to be tested shall be described. To begin with, a recap of the workings of the ILS algorithm will be provided. Following this, a new *Adaptive Memetic Algorithm* will be proposed and described. Also described shall be the variant of the memetic algorithm which makes use of the solution distance measure.

5.3.1 Adaptive Iterated Local Search

The algorithm to be tested here is the *Adaptive Iterated Local Search* described in the previous chapter (see section 4.4.1) which includes adaptive techniques for selection of both the mutation and local search heuristics. The workings of the algorithm are described there in detail but are summarised briefly below.

- **Repeat the following**
  - **Mutation** An *Extreme Value-based Adaptive Operator Selection* mechanism is used to select a single *Mutation* heuristic to be applied based upon how they have previously impacted objective function values. This heuristic is applied to the incumbent solution s to create a working solution s_0.

  - **Local Search** During this stage, the *Local Search* heuristics are initially ordered using a measure of their average performance over previous applications. Using this ordering, they are then applied in turn to the working solution, s_0, until an application does not yield an improvement in objective function value.
  - **Solution Acceptance** The new solution, s_0, is accepted if its objective function value is superior (lower) than that of the solution before the mutation stage (s).

- **Until time limit has been reached.**

5.3.2 Adaptive Memetic Algorithm

In order to demonstrate the potential the HyFlex framework has to accommodate population-based algorithms, methods from the field of *Adaptive Memetic Algorithms*[119] have been selected. Rather than using a completely different branch of algorithms, it would be more accurate to state that the memetic algorithms to be proposed are an extension of the ILS algorithm described previously. These algorithms follow the same basic structure as ILS, with the additions of a population and the use of crossover heuristics. The choice of algorithm is inspired by the work of the previous chapter, which showed that a simple framework with few parameters can effectively adapt to differing problems. The use of adaptive selection mechanisms will remain and is the key element that allows these algorithms to be considered hyper-heuristics. Below, the separate stages of the *Adaptive Memetic Algorithm* will be described. Firstly, though, it must be explained that the algorithm contains a population of size j. The first stage of the algorithm is to initialise all members of the population after which a loop is entered. This loop continues until the time limit for the run is met. Preliminary experiments with a small set of possible population sizes were performed and indicated that a population size of 4 provided a sufficient opportunity for solution diversity without taking up large amounts of resources at each iteration of the loop.

Crossover Heuristic Application

The crossover stage of this algorithm could be described as a greedy method. A greedy mechanism is selected as the population size of 4 is low and computation time will not be affected to a great extent. As mentioned in the initial HyFlex description, each crossover heuristic receives two solution indices as input. These could be described as parent solutions. In this crossover stage, for each solution in the population, the application of a crossover heuristic with every other solution is considered. As an example, for this algorithm where there is a population size of 4, the current working solution separately applies a crossover heuristic with itself and each of the remaining 3 solutions as inputs. In all current HyFlex domain, the number of available crossover heuristics is low (¡=2). Therefore a sophisticated selection mechanism would be somewhat wasted on this stage of the algorithm. For that reason, for each of these independent applications, a crossover heuristic is chosen uniformly at random to be applied.

The objective function values for the solution after each heuristic application are stored. After all applications of a crossover heuristic for a particular solution, the resulting solution with the lowest(i.e. superior) objective function score is selected as the solution to be kept. The entire procedure is described in Algorithm 15.

Algorithm 15 The *crossover heuristic stage* sub-routine for the memetic algorithm which takes as input the set of solutions, S, in the population.

```
procedure CROSSOVER STAGE($S$)
    for all Solution $s$ in $S$ do
        for all Solution $s'$ in $S$ do
            if $s! = s'$ then                          ▷ If not the same solution
                Solution $s'' \leftarrow applyRandomCrossoverHeuristic(s, s')$
                if $objFunc(s'') < objFunc(s)$ then
                    $s \leftarrow s''$
                end if
            end if
        end for
    end for
end procedure
```

Mutation Heuristic Selection

Following the crossover stage of the algorithm, a loop of the population members is entered. During this loop, each population member is the subject of an application of a mutation heuristic, followed by the local search stage which includes multiple applications of local search heuristics. Both of these stages use the same means of rewarding heuristics, but differ slightly in their structures. A conclusion drawn from the previous chapter was that simple methods of selecting heuristics allowed a high level of adaptability to different problems. For that reason, a simple method is used here also. Further, it is a deliberately simplistic method in order to allow basic examination of the feasibility of basic population-based methodologies for the HyFlex domain. In addition, the simplistic measure allows for speed of execution, permitting greater numbers of iterations to be performed.

The basic method of reward is that a heuristic which yields an improvement in objective function score receives a single point. A single point is used so that an operator receives some reward for improving a solution but not so much that it can overpower other heuristics.

It is important to emphasise a particular point. The application of a mutation heuristic is not necessarily expected to yield an improvement in objective function value, but instead to change or mutate the solution. Thus, it would be meaningless to reward a mutation heuristic for its ability or lack thereof to immediately improve a solution. Instead, the mutation heuristics are, in this algorithm (a variant will be proposed later in this chapter that uses solution distance as a performance measure), judged on whether an improvement in objective function value is found after *both* the mutation and local search stages. In such a way, the aim is to reward mutation heuristics that modify solutions in a fashion that provides them with the potential to improve.

At each iteration, a single mutation heuristic is selected to be applied by using the scores of all mutation heuristics. The previously mentioned *Roulette Wheel Selection* mechanism (see Algorithm 13 in Chapter 4) is used to translate the scores into a choice of heuristic. This mechanism works by assigning each heuristic a 'chunk' of a figurative roulette wheel, with a size proportional to that heuristic's score in relation to the total scores across all heuristics. A random number generator is then used to navigate around the roulette wheel. The heuristic located at the relevant point of the roulette wheel is selected to be applied.

The final point of interest regarding this scoring and selection mechanism is that the score for a heuristic must have an initial seed value. If any heuristic started with a value of 0, it could never be selected under the roulette wheel selection mechanism. For the mutation heuristics, the initial value is set as 1, so that there is an equal probability of selecting each heuristic to begin with. The value is set low as the mutation heuristics are called infrequently and so low values are required to achieve meaningful results.

Local Search Stage

The local search stage of this algorithm uses the same scoring system for heuristics as the mutation stage for the same reasons of simplicity and desire for less use of parameters. However, there are subtle differences in the implementation. The overriding reason for these differences is the differing way the stages operate. It was explained in the previous section that a single mutation heuristic is selected and applied at each iteration. The local search stage contains many more applications of heuristics within a single iteration as the over-riding goal is to improve a solution, not mutate a solution. The mutation can be necessary to enable improvement but when applied too often can hinder improvement.

The goal of the stage is to ensure that a local optimum has been reached. To this end, a tracker keeps stock of how many local search heuristic applications have occurred since the last improvement in objective function score. Local search heuristics are repeatedly selected by the roulette wheel mechanism, then applied to the solution until the limit (*maxIterations*) of non-improving applications has been reached. Following preliminary testing of a variety of values, a value for *maxIterations* of 40 has been used, to provide a balance between allowing sufficient time for improvements to be found, and not using a surplus of processing time. The seed score given to each of the heuristics is 100 for the local search stage. This is because there are many more applications of local search heuristics than mutation heuristics and if the seed value is not sufficiently high then it is easy for one heuristic to dominate the selection. The values proposed here appear to work well for the instances tested. However, should the set of instances be expanded, it could be that they are no longer appropriate. Further investigation could be undertaken to establish the sensitivity of the values. Pseudocode for the local search stage can be seen in Algorithm 16.

Algorithm 16 The *local search stage* sub-routine for the memetic algorithm which takes as input the a solution, *s*, in the population and the set of *scores*, *ScoreLS* for the local search heuristics.

```
procedure LOCAL SEARCH STAGE($s, ScoreLS$)
    $intsSinceImprovement \leftarrow 0$
    $iterationLimit \leftarrow 40$
    while $intsSinceImprovement < iterationLimit$ do
        LSHeuristic $i \leftarrow rouletteSelection(ScoreLS)$
        $s' \leftarrow applyHeuristic(i, s)$
        if $objFunc(s') < objFunc(s)$ then
            $s \leftarrow s'$
            $scoreLS_i \leftarrow scoreLS_i + 1$
        end if
    end while
end procedure
```

Overview of Algorithm

The previous sections have described the precise details of the main stages of the *Adaptive Memetic Algorithm*. This section will outline how the individual stages fit in to the algorithm as a whole. As explained previously, the algorithm follows the same basic structure as for *Iterated Local Search*, that of a mutation, followed by a local search stage and a solution acceptance decision. The structure here is the same with the addition of a population, and a crossover stage. The entire process is shown in Algorithm 17.

Algorithm 17 A memetic algorithm for use in HyFlex.

```
procedure MEMETIC ALGORITHM
    $popSize \leftarrow 4$                                    ▷ Population size
    for $i \leftarrow 0, popSize$ do
        Solution $s_i \leftarrow initialiseSolution(i)$
    end for
    for $i \leftarrow 0, noOfMutHeuristics$ do       ▷ For the no. of mutation heuristics
        $scoreMut_i \leftarrow 1$                          ▷ The ‘score’ for mut heuristic $i$
    end for
    for $i \leftarrow 0, noOfLSHeuristics$ do    ▷ For the no. of local search heuristics
        $scoreLS_i \leftarrow 100$                          ▷ The ‘score’ for LS heuristic $i$
    end for
    while $timeHasNotExpired()$ do
        for $i \leftarrow 0, popSize$ do
            $s_i \leftarrow crossoverStage(S)$
            Solution $tempSol \leftarrow s_i$
            $mutHIndex \leftarrow rouletteSelection(ScoreMut)$
            $tempSol \leftarrow applyHeuristic(mutHIndex, tempSol)$
            $tempSol \leftarrow localSearchStage(tempSol, ScoreLS)$
            if $objFunc(tempSol) < objFunc(s_i)$ then
                $s_i \leftarrow tempSol$
                $scoreMut_mutHIndex \leftarrow scoreMut_mutHindex + 1$
            end if
        end for
    end while
end procedure
```

5.3.3 Diversity Variant

The above section has outlined the basic workings of the *Adaptive Memetic Algorithm*. For the memetic algorithm, a simple reinforcement learning technique has been utilised to reward successful heuristics. For the ILS described in the previous chapter, an *Adaptive Operator Selection* technique has been applied. In this section, a more sophisticated technique will be described which not only considers any improvement in objective function value, but also considers to what extent a mutation heuristic can modify a solution. To judge the extent to which a solution has been modified, the newly introduced diversity function of HyFlex can be used. This method will be used for both the mutation and local search stages. The idea to consider both diversity and objective function improvement was proposed initially in [103]; however the method to be used here was proposed first as the 'Compass' mechanism in [104]. In the [104] paper, vectors are used which represent objective function improvement and diversity along with a vector representing the direction of the search. In other words, this final vector controls the balance between exploration and exploitation in the search.

The actual implementation used here differs from that used for 'Compass'. A simplified method has been used in order to work better toward the goals of robustness and flexibility that a hyper-heuristic should demonstrate. In keeping with the findings of the previous chapter, only a single parameter is used for this version of the algorithm. The method works by recording the mean objective function improvement and mean diversity value over all applications of a heuristic. To calculate a score for a heuristic, which can be used for heuristic selection, a parameter c is used. This parameter represents the balance between the importance of objective function improvement and solution diversity for a heuristic and is a value between 0 and 1. To show the workings explicitly, the following formula is used to calculate the score for a heuristic.

$$score = c * meanObjImprovement + (1-c) * meanDiversity$$

Once these scores have been calculated for all operators, then the previously described *Roulette Wheel Selection* mechanism is used to select a heuristic to be applied. For the mutation heuristics, a value for c of 0.1 was selected, to ensure that the heuristics were judged almost entirely on their ability to mutate a solution, with some small regard paid to their potential to improve on objective function score following the local search stage. For the local search stage, a c value of 0.9 was selected. This value was selected in order that the focus for this set of heuristics would be on improving objective function value for a solution. The parameter value is set at a little under 1 as it may be advantageous for the local search heuristics to show the ability to mutate the solution a little. For example, a heuristic may modify a solution in such a way that the objective function score remains the same but the actual solution differs genetically from the previous solution. In this sort of circumstance, it may then be easier in the future to find gains.

5.4 Experiments

5.4.1 Instances

All testing will be performed on the *Vehicle Routing Problem with Time Windows (VRPTW)* domain as the new features of HyFlex are currently only available for this domain. The same instances will be used that were used for the work in the previous chapter. That is, 5 instances from the Solomon dataset, and 5 from the Gehring and Homberger dataset. The table from the previous chapter is reproduced here (table 5.1) for ease of reading. These instances have been selected to represent a range of instances within the VRPTW.

Instance	Name	No. Vehicles	Vehicle Cap.	No. Custs.
0	Solomon/RC/RC207	25	1000	100
1	Solomon/R/R101	25	200	100
2	Solomon/RC/RC103	25	200	100
3	Solomon/R/R201	25	1000	100
4	Solomon/R/R106	25	200	100
5	Homberger/C/C1-10-1	250	200	1000
6	Homberger/RC/RC2-10-1	250	1000	1000
7	Homberger/R/R1-10-1	250	200	1000
8	Homberger/C/C1-10-8	250	200	1000
9	Homberger/RC/RC1-10-5	250	200	1000

Table 5.1: A table showing the VRPTW instances to be used for testing.

5.4.2 Test Details

This section shall consider only the algorithms mentioned above and comparisons between those algorithms as well as to best-known solutions. A separate section shall consider the results of the CHeSC hyper-heuristic competition and how these algorithms would perform within that. For these tests, run-times of 20 CPU minutes will be used with 10 runs to be made per instance for all 10 instances shown above. This longer run-time compared to previous tests will ensure that the full potential of the algorithms will be demonstrated. A machine with a 2.27GHz Intel(R) Core(TM) i3 CPU and 4GB RAM will be used to run the tests. A total of 4 algorithms will be tested here (or rather, two algorithms, each with variants) and the details of these can be seen below.

- *AILS* The most successful *Iterated Local Search* algorithm described in the previous chapter, with an *Adaptive Operator Selection* mechanism used to select mutation heuristics and an ordered local search stage.

- *AILS-C* The same algorithm as above, with the exception that the 'Compass' mechanism described in section 5.3.3 is used to score and select mutation heuristics.
- *AMA* The *Adaptive Memetic Algorithm* described in section 5.3.2 with a population of solutions and a crossover stage. A reinforcement learning technique is used to score and select both the mutation and local search heuristics.
- *AMA-C* The same algorithm as above with the exception that the 'Compass' mechanism described in section 5.3.3 is used to score and select heuristics for both the mutation and local search stages.

5.5 Results

Four forms of analysis of results shall be performed for this work. The first is ordinal data analysis and will use the Borda count method mentioned in the previous chapter to compare the performance of the algorithms over the 10 instances as a whole. Secondly, analysis shall take place on the distribution of objective function values for individual instances. This will be done with the use of box-plots which can represent performance over a number of runs in an easy-to-interpret fashion. Thirdly, comparisons shall be made to the best-known results for the 10 instances tested. The final form of analysis will determine whether there is statistical significance between the results.

5.5.1 Ordinal Data Analysis

For this form of analysis, the *Borda Count* method will be used. As was described in the previous chapter, the *Borda Count* method operates by assigning ranks to each algorithm for each instance and then calculating a score for an algorithm as the sum of their ranks over all instances. As an example, an algorithm that ranked first in all 10 instances, would have a *Borda* score of 10. As there are 4 algorithms and 10 instances, the worst possible score for these tests is 40(4*10).The measure of performance used in calculating the score is the median value across the 10 runs. The chart in figure 5.1 shows the results for this testing.

As can clearly be seen from the figure, the *Adaptive Memetic* algorithms strongly outperform the *Iterated Local Search* algorithms over the 10 instances. This gulf in performance is not replicated when considering the benefits of the 'Compass' variants of the algorithms. For the memetic algorithms, there is no difference at all in *Borda* score whereas for the ILS algorithms the 'Compass'

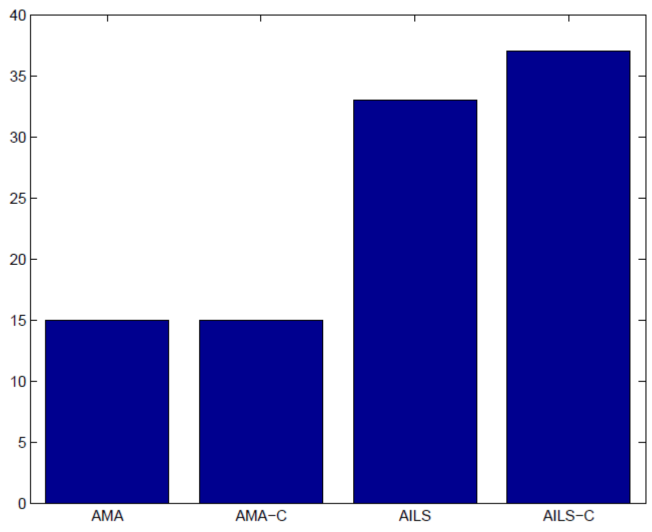


Figure 5.1: A graph showing the total *Borda Count* scores for the 4 algorithms across the 10 instances tested.

variant actually performs worse than the original version, with 37 and 33 points respectively.

5.5.2 Distribution of Objective Function Values

Box-plots can be useful tools for analysis as they demonstrate results for an algorithm across all runs for an instance. This information can be viewed in order to determine how tightly bunched the results are - in other words how consistent the algorithm is. They can also be useful in demonstrating different patterns between instances and indeed the scale of differences between algorithms. Box-plots will now be shown which demonstrate the results for all of the harder Gehring and Homberger instances. Figures 5.2, 5.3, 5.4, 5.5 and 5.6 show the distribution of objective function values for the 10 runs for each algorithm on instances 5, 6, 7, 8 and 9 respectively.

The results from the ordinal data analysis in the previous section are also borne out in the box-plots of these instances. The gap in performance between the two forms of algorithm are visually obvious with the greatest distinction being observed in instances 5, 6 and 8. In all instances, it is the AMA algorithms which are far stronger than the ILS methods on all statistical indicators, i.e. median and upper and lower quartiles. As well as obtaining stronger results in terms of objective function value, the AMA algorithms demonstrate a greater consistency of result over all the instances shown. The range of the ‘boxes’ is far smaller than for the ILS algorithms, indicating not only that the memetic methods can produce strong one-off results, but that they are able to repeat the performance

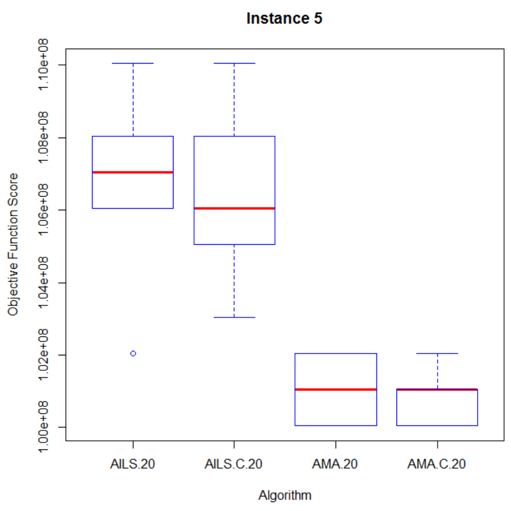


Figure 5.2: A box-plot showing the distribution of objective function values for instance 5.

over several runs. This quality of robustness is particularly important for these hyper-heuristics which have the explicitly stated aim of providing robust and flexible performance across multiple runs of multiple instances of varying problems. While the differences noted are relative to the objective function scores of the tested algorithms, testing to follow in the next section will help establish the performance of the algorithms in comparison to the best-known results from the literature.

A further point of interest is the performance of the variants that use the newly added distance measure. Whilst the median values of the non-distance and distance variants may be similar, there is a plain difference evident in the diagrams of a stronger 'lower end' performance from the distance variants. There are more results at the median level and below for the variants using the distance measure. It must be remembered that this is the first time a distance measure has been used as a performance metric in a hyper-heuristic. Therefore, these results are very much preliminary. For the method to show some increase in solution quality at this early stage demonstrates a potential for distance to be used in different hyper-heuristic methods. The winner of the CHeSC competition, *AdapHH* [110], combines multiple performance measurements when selecting low-level heuristic. This may be a strong algorithmic framework in which to apply a distance measure.

5.5.3 Comparison to Best-Known Results

From the previous sections, it would seem that the *Adaptive Memetic Algorithm* shows a stronger performance than the ILS under these test conditions. Now,

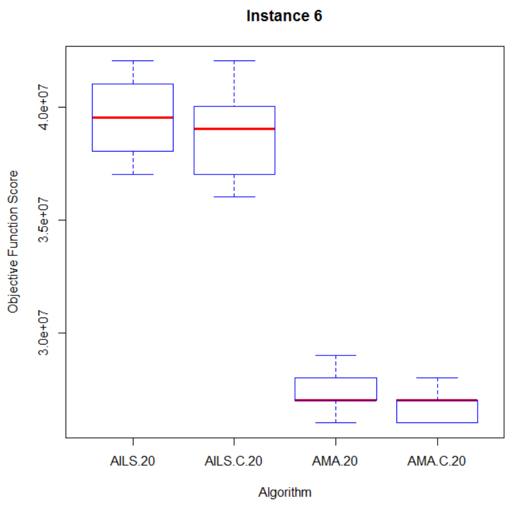


Figure 5.3: A box-plot showing the distribution of objective function values for instance 6.

results must be considered in the context of all work done on these problem instances. The simplest way to achieve this is to compare the results obtained by the best-performing of the algorithms tested (the memetic algorithm) to the best-known results for the 10 instances. As the VRPTW is a dual objective problem with the primary objective being minimisation of the number of routes and the secondary objective being distance reduction, the results will be considered in this way also. Table 5.2 shows the best results obtained for each instance by the two variants of the *Adaptive Memetic Algorithm* over the 10 runs per instance, as well as the current best-known results.

Instance	No. of Vehicles			Distance		
Name	*AMA*	*AMA-C*	*Best-k*	*AMA*	*AMA-C*	*Best-k*
0-SRC207	4	**3**	**3**	**1047.42**	1133.83	1061.14
1-SR101	**19**	**19**	**19**	1650.8	**1631.82**	1645.79
2-SRC103	**11**	**11**	**11**	1276.82	1263.78	**1261.67**
3-SR201	**4**	**4**	**4**	1261.043	1276.45	**1252.37**
4-R106	**12**	**12**	**12**	1268.93	1284.23	**1251.98**
5-HC1-10-1	**100**	**100**	**100**	42481.26	42485.04	**42478.95**
6-HRC2-10-1	26	26	**20**	33272.57	**32839.49**	63373.15
7-HR1-10-1	**100**	**100**	**100**	59020.74	60517.21	**53904.23**
8-HC1-10-8	101	101	**93**	44037.96	44120.54	**42499.59**
9-HRC1-10-5	94	93	**90**	52581.52	52439.09	**46631.89**

Table 5.2: A table showing a comparison of the *Adaptive Memetic Algorithm* best results to the best-known results.

In the previous sections, analysis of the results has indicated that there is some difference to be found between the standard *Adaptive Memetic Algorithm* and the

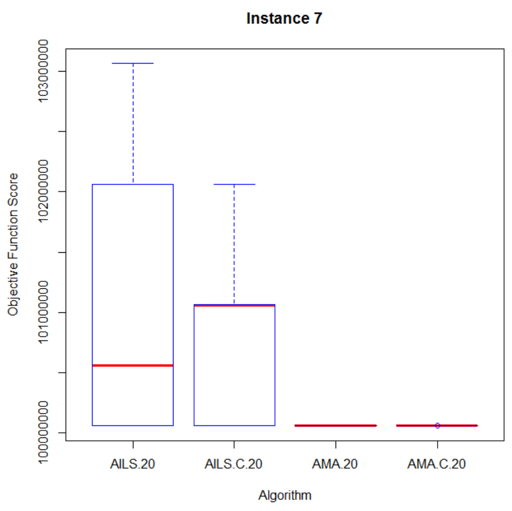


Figure 5.4: A box-plot showing the distribution of objective function values for instance 7.

variant of the algorithm that uses the 'Compass' method; although the scale of this difference is not large. The data provided in table 5.2 paints a slightly different picture. Examining first the primary objective, the number of vehicles, it can be seen that the *AMA-C* method matches the basic *AMA* on 8 instances and betters it on 2 instances. Hence, it would seem that using a distance measure in the context of this algorithm provides an advantage when considering the best results obtained over a number of runs, even if the median results do not greatly differ from the standard method. Examining the distances is less meaningful as improving the primary objective of distance can cause an increase in the distance travelled. Of the instances where both algorithms achieve the same number of routes, the basic *AMA* method obtains a lower distance in 5 out of 8 instances, implying that the *AMA-C* variant is less efficient at reducing the distance travelled. However, the ability to improve the primary objective of a solution shows a real difference can be made by using a distance measure.

The next element to be examined is performance of the algorithms compared to the current best-known solutions for these 10 instances. For the first 5 instances, i.e. the Solomon instances, the *AMA-C* algorithm matches the best-known solutions in terms of routes. When considering the distance travelled, the *AMA-C* method actually achieves a new best-known solution for one instance (*SR101*). This is a welcome achievement for an algorithm that has more of an emphasis on being robust and flexible than on achieving strong individual results. For the Gehring and Homberger instances, the results are more mixed. On two of the instances, the *AMA-C* algorithm matches the number of routes with the best-

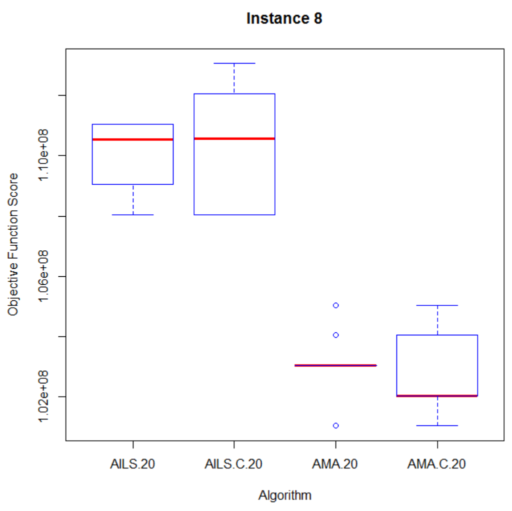


Figure 5.5: A box-plot showing the distribution of objective function values for instance 8.

known solutions. For the other instances, the difference are 6, 8 and 3 routes for instances 6, 8 and 9 respectively. The difficulty in judging performance by referencing distance travelled when the number of routes is different is evident for instance 6. For this instance, the *AMA* algorithms both use 6 more routes than the best-known. For the distance travelled, however, they almost half the amount of the best-known. This raises an interesting question about how to assess the worth of a solution. In real-world applications, it might be the case that the distance travelled for the best-known is so high that it over-powers the benefits gained from the reduction in routes. Hence, a weighted sum objective function may be of more use for commercial applications.

5.6 CHeSC Competition Analysis

The *Cross-Domain Heuristic Search Challenge (CHeSC)* which took place in 2011, was the first competition to use the HyFlex framework. It was an international research competition which had the aim of promoting research into algorithms that demonstrate good general performance on several problem domains and instances. This section shall briefly describe the format and workings of the competition, before summarising the best-performing algorithms and, finally, comparing the best competition results to those obtained by the *Adaptive Memetic* algorithm described previously within this chapter of the thesis.

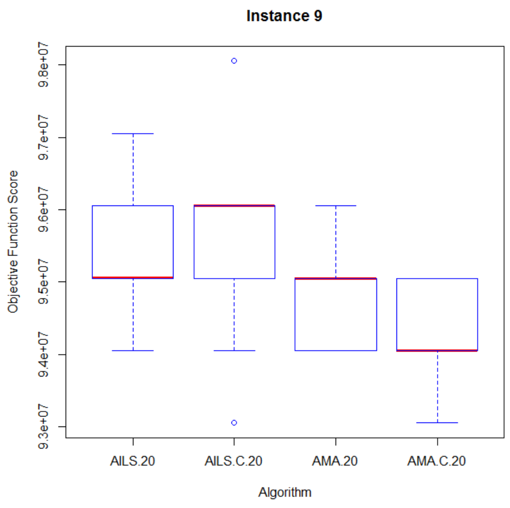


Figure 5.6: A box-plot showing the distribution of objective function values for instance 9.

5.6.1 Competition Format and Rules

All competitors were required to submit a single hyper-heuristic in the form of a java file. This heuristic would then be tested on 4 'seen' problem domains and 2 'hidden' problem domains. The seen domains were *Personnel Scheduling*, *Permutation Flow Shop*, *One-dimensional Bin Packing* and *Maximum-Satisfiability*, in other words the original 4 HyFlex domains. From these 4 domains, only a handful of sample instances were available for testing in advance, with the majority of instances used for final testing not having been previously seen by the competitors. This helped ensure that algorithms could not be tuned to work particularly well for these problems and instances. This cause was further aided by the use of 2 hidden domains, the *Vehicle Routing Problem with Time Windows* and *Travelling Salesman Problem* domains. The VRPTW domain was the one described previously within this thesis. For all of the domains, including seen and hidden domains, the algorithms were tested on 5 instances, selected by the organisers and not known in advance. For each of these instances, 31 runs were performed for each algorithm at a run time of 10 CPU minutes per run. For each instance, the median result from the 31 runs was taken and used as the measure with which to calculate an algorithm's score. Scoring operated using a means of ordinal data analysis, a type of scoring that provides meaningful results even when the scales of objective function values differ greatly between problems. The actual method used is comparable to the scoring system for the *Formula 1 World Championship*, with scores of 10, 8, 6, 5, 4, 3, 2 and 1 being rewarded to algorithms ranking 1st, 2nd, 3rd, 4th, 5th, 6th, 7th and 8th respectively. The winner of the competition was the algorithm with the highest aggregate points score across all 6 instances.

5.6.2 Results and Best Algorithms

The full results from the competition can be seen at [115]. A chart summarising the results for the top 4 competitors can be seen in figure 5.7. A brief description of each of the top 4 algorithms can be found below.

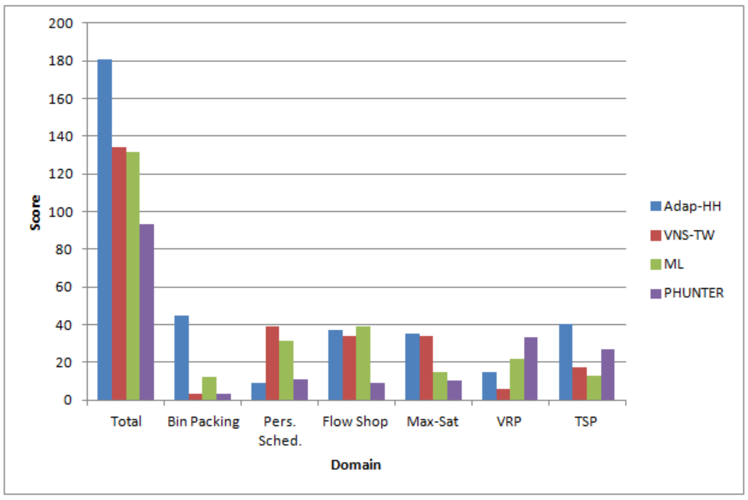


Figure 5.7: A chart showing the scores for the top competitors across each domain and in general for the CHeSC 2011 competition.

- *Adap-HH* The Misir algorithm, as described in [110], includes two main stages of heuristic selection and solution acceptance. During heuristic selection, both dynamic heuristic sets and pairs of heuristics are considered in order to provide the best combination of heuristic applications. The parameters used by heuristics in HyFlex, *intensityOfMutation* and *depthOfSearch*, are also adaptively modified. For the solution acceptance element of the algorithm, an adaptive threshold acceptance mechanism is used. The combination of these various adaptive elements helped *Adap-HH* to the strong cross-domain performance that led to an overall victory in the competition.

- *VNS-TW* The approach presented in [82] is a *Variable Neighbourhood Search* that includes a diversification and an intensification stage. An ordering is given to the mutation heuristics and an adaptive method is used to determine the value of the *depthOfSearch* parameter.

- *ML* An approach very similar to the *Adaptive Iterated Local Search* method described previously. A reinforcement learning technique is utilised in heuristic selection and a simple adaptive measure used for solution acceptance.

- *PHUNTER* Once again this algorithm follows the basic ILS structure but differentiates itself by combining different ILS techniques. The method uses the metaphor of pearl-hunting to describe the stages of diversification and intensification. A full description of the algorithm can be found in [32].

It is evident from the graph that the *Adap-HH* algorithm is the significantly stronger competitor across the domains as a whole. In addition, it achieves outright victory in 2 domains, showing the potential for strong individual performances as well as a high level of consistency. However, the results also show a level of difference between some domains, implying that the approach does indeed work better for some problems than others. This comes back to the idea of the 'Free Lunch Theorem' [169] which states that any 2 algorithms would end up with an indentical mean objective function value if used on all possible optimisation problems. In other words, there does not exist a 'universal' algorithm, an algorithm that could solve to optimality all possible problems. Even for the extremely small subset of possible problems used for the CHeSC competition, it can be observed that the winning algorithm does not perform best on all problems or all instances of a problem. However, this does not detract from the idea of an algorithm that can adapt and perform 'reasonably well' for different problems. Indeed, the motivation behind development of general and robust algorithms is often to achieve performance within a certain percentage of best-known solution performance for a variety of problems/instances, rather than to achieve perfection for every problem encountered.

5.6.3 Experiments and Results

The top results from the competition have been given and the competitors acheiving these results have been described. Now, these top algorithms will be compared to the hyper-heuristic algorithms previously proposed in this chapter. Specifically, the *Adaptive Iterated Local Search(AILS)* algorithm and the *Adaptive Memetic Algorithm(AMA)* will be included in the comparisons, along with the top 3 competition algorithms in the VRPTW domain, *PHUNTER*, *HAEA* and *KSATS*. The variants of the algorithms which use the 'Compass' mechanism can not be included in these comparisons as the version of HyFlex used for the CHeSC competition did not include the new features proposed above, including the genotypic distance measure. In addition, further testing of the *AILS* and *AMA* algorithms had to be performed in order to mirror the conditions of the testing for the CHeSC competitors. That is to say, 31 runs of 10 CPU minutes length for the 5 instances of the VRPTW domain.

The method used to compare these algorithms will be the *Borda Count* method, used in the same fashion as in previous sections and chapters. As there is only a single domain and 5 instances to be tested, the minimum (i.e. best) possible score is 5 and the worst possible is 25 (as there are 5 algorithms). The graph in 5.8 shows the *Borda Count* scores as a bar chart.

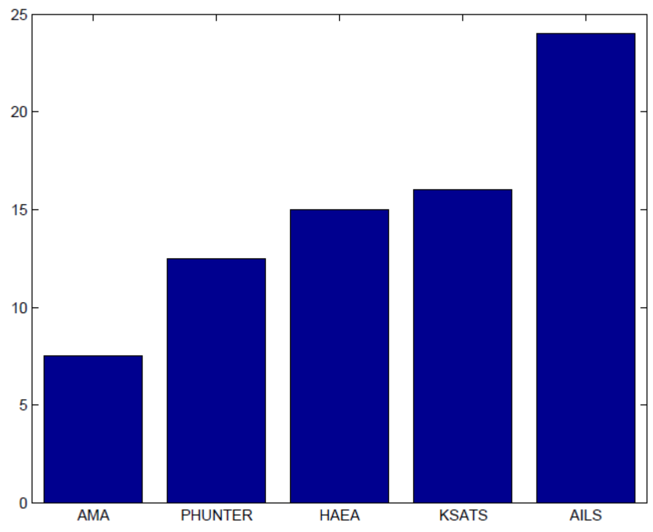


Figure 5.8: A bar chart showing the *Borda Count* scores for the 5 algorithms across the 5 instances.

The results show that the *Adaptive Memetic Algorithm* is comfortably stronger than the top competitors from CHeSC. This can be considered a strong result given the quality of algorithms that were tested. The AMA method also comfortably beats the method of Misir et al. [110] on the VRPTW. The Misir et al. algorithm won the CHeSC competition of 2011. The main qualifier to these results is that the CHeSC algorithms were designed for and tested on 6 problem domains, whereas the testing of the AMA was only performed on the VRPTW problem domain. However, it can still be understood that AMA produced a consistent performance across the many runs for the 5 instances indicating a robustness to the algorithm. Furthermore the concept of generality and flexibility doesn't have to be considered only in a cross-domain context. In a real-world application, it may be desirable for algorithms to be able to adapt to changing situations. However, the underlying problem is still essentially the same. The changes to be adapted to may be in the form of a new constraint or a change in data. Therefore, to show strong, consistent performance on a set of different problem instances within the same basic problem can still show generality and adaptability.

In contrast to the AMA method, the *Adaptive Iterated Local Search* algorithm produced poorer results, placing last out of the tested algorithms. When considering the results from this chapter in general, the main point of interest is why the memetic algorithm produces results that are so much stronger than the ILS algorithm when they share the same basic structure. The first possibility is that the introduction of a population allows greater diversification within the search, as there are more solutions and, hence, a greater level of opportunity to be in different areas of the search space. A second possibility is that the use of the crossover heuristics provides a significant advantage for this VRPTW domain. As stated in the description of the domain in the 2nd chapter, the crossover heuristics used are newly proposed and are previously untested. The strength of these heuristics in comparison to standard literature methods may merit further investigation.

5.7 Conclusion

This chapter has introduced several additions to the VRPTW HyFlex domain, all designed to provide more tools or information to a hyper-heuristic. There has been a question of whether providing this extra information can allow a hyper-heuristic to achieve improved results. To address this question, motivations have been provided behind the design decision for each new component, with explanations given of how a hyper-heuristic could use the new features. A practical example has been given of how one of the new components, the measure of distance between solutions, could be used by a hyper-heuristic. In presenting a population-based *Iterative Local Search*, it was shown how the distance measure could be used to inform selection of mutation heuristics by the hyper-heuristic. It was shown that using this method could also provide improved results, with a hyper-heuristic that used the distance measure yielding lower 'best' objective function values than a similar hyper-heuristic which didn't use the distance measure. This is a significant contribution which demonstrates the potential of using a distance measure in the context of a hyper-heuristic.

The other element of domain design considered within this chapter was the differences between single-point and population-based algorithms regarding how they interact with problem domains. The algorithm above was described in detail in the chapter, with design decisions being explained with reference to the domain. Having a population-based approach allowed the hyper-heuristic to access more tools of the problem domain (e.g. the crossover heuristics) which benefitted it, as demonstrated through vastly superior result to that of the similar single-point hyper-heuristic.

Chapter 6

A General Domain for the Vehicle Routing Problem

6.1 Introduction

The work presented in the thesis so far had examined the relationship between a problem domain (particularly Vehicle Routing Problem domains) and the hyper-heuristics that use it. The design of the many components of a problem domain have been considered, as well as the way in which extra information provided by a problem domain can be utilised by the hyper-heuristic. The problem domain that has received most focus within this thesis has been the Vehicle Routing Problem with Time Windows. This a a problem that has many practical, real-world applications. For example, large delivery companies with significant fleets could save a substantial amount of money by reducing the distance travelled by its vehicles, or by reducing the number of vehicles needed. These types of industrial applications will often be far more complex than the academic problems, with an increase in constraints and performance metrics. Thus, the design of a real-world domain may need to be different to before. The question to be answered here is 'What qualities are required by a problem domain for it to be suitable for real-world problems?'.

This chapter will address this question by proposing a new Vehicle Routing Problem domain that allows representation of problems more akin to real-world routing problems. The design decisions will be explored, with reference to what is needed to address the complexities of these types of problems. One particular issue will be considered, to demonstrate one factor that may be relevant to such a problem. This is the issue of fairness between routes, i.e. to attempt to achieve a balance between the number of customers served by each route, or the distance travelled by a route. This is a consideration in industrial applications where it would be undesirable to have an employee shift which consisted of only a single customer visit.

6.2 Motivation

In chapter 2.2.2, the basic *Vehicle Routing Problem(VRP)* is described; that of satisfying a set of customer with a fleet of vehicles demands whilst minimising the number of vehicles needed and the distance travelled. The most widely studied academic problems were described, namely the *Capacitated Vehicle Routing Problem (CVRP)* and the *Vehicle Routing Problem with Time Windows (VRPTW).* The CVRP adds a constraint where the vehicles have a capacity which can't be exceeded. The VRPTW adds a time window constraint where each customer has a time window in between which their service must begin. Both of these constraints represent elements of real-life vehicle routing applications; however real-world problems often include many other features and constraints that aren't currently represented within academic works. There are many examples of real-world problems being modelled and solved in singularity. Examples of a wide range of routing problems from different industries are given in [72]. The problem, though, can be that it is often the case that algorithms are often designed only for use with a single problem and can't be easily adapted when constraints change. The domain presented here will allow for a wide range of VRPs to be represented, without any need to change the domain.

One real-world routing feature in particular shall be used to demonstrate the utility of the new features of this domain. This is the concept of 'fairness' between routes within a solution; the issue of ensuring that no single route has too high or too low a workload. There are several practical motivations for this. Firstly, a short route can cause problems with respect to the employee who will drive the vehicle. The company may be required to pay the employee for a full day even if their workload doesn't amount to a full days work which is undesirable from an economic efficiency viewpoint. On the other hand, if the work id paid by the hour, the company may find it difficult to find staff to work very short shifts. Furthermore, there is the issue of vehicle maintenance. A vehicle with a high load and which is travelling long distances will experience more wear than a vehicle with shorter routes and a lighter load. In that case, it would be beneficial to have a more balanced distribution of work load in order that certain vehicles don't wear out quickly, resulting in large repair costs for the company.

However, it is unlikely that a 'fair' solution will always be an optimal solution. Often, when customers are in clusters, the optimal solution will be a solution where each vehicle services one of these clusters of customers. Where these clusters vary in size, a solution of this nature can be a very 'unfair' solution. While a company may want fairer solutions for the reasons mentioned above, using much worse quality solutions because they are fairer will bring unwanted extra costs. Herein lies the crux of the fairness problem - can solutions be made fairer without significantly impacting on solution quality? The following sections will demonstrate how the additional features added to this domain can be used to investigate the issue of fairness.

6.3 Definition of Fairness for Initial HyFlex Domains

In the sections following this, the concept of fairness will be covered in detail for the *Vehicle Routing Problem*, including a description of an implementation. However, to understand the benefits of this kind of tool, fairness will also be discussed here for the original 4 HyFlex domains - *Permutation Flow Shop*, *Personnel Scheduling*, *One-dimensional Bin Packing* and *Maximum Satisfiability (MAX-SAT)*. This will contribute to understanding of how seemingly simple problems and representations can yield significant data to a hyper-heuristic.

6.3.1 Permutation Flow Shop

It would initially seem difficult to define a measure of fairness on the *Permutation Flow Shop* problem. All jobs must be processed on all machines so, in any solution for an instance, all machines will have processed the same load. One way fairness could be measured would be to consider idle times of machines. For example, one machine may stand idle for an amount of time at the start and then have a continuos heavy load later on whereas another machine might have a steady flow of jobs to be processed throughout. Fairness in this sense could be encouraged through an objective function term which measured the idle times of machines and attempted to keep idle times within specified minimum values. The same theory of fairness could perhaps be applied to the jobs to be processed. As an example, a job might have a lot of idle time if waiting for another job to finish processing. However, by minimising the standard objective function of the makespan, this idle time will inevitable be driven down anyway and so extra measure would be superfluous in this instance.

6.3.2 Personnel Scheduling

With the *Personnel Scheduling* domain, it is clearer to see how a solution could be considered 'fairer'. For example, it may be 'unfair' to have an employee with multiple shifts in a short period of time. Another example might be the time of day or period of the week in which an employee has shifts. There is a difficulty though. In this domain, there are many different problem variations with many constraints. This makes it difficult to establish a single measure of fairness that covers all possibilities. One suggestion to represent fairness in this domain may be for an instance to specify fairness criteria, which may span several factors. This follows what already happens in this domain, where constraints are represented as objective function terms. The given fairness criteria could then be calculated for each employee and used to drive fairness.

6.3.3 One-dimensional Bin Packing

For the *One-dimensional Bin Packing* problem domain, it is very simple to achieve a simple definition of fairness. This is because fairness is already being driven from the objective function of the domain. The objective function currently calculates the average 'fullness' of a bin (see [86] for more details). By trying to balance fullness, an attempt to have a 'fair' distribution of items is present.

6.3.4 Maximum Satisfiability (MAX-SAT)

The final domain, representing the *Maximum Satisfiability* problem, demonstrates that the concept of 'fairness' may not be applicable to all problems. As the problem is concerned with maximising the number of 'satisfied' clauses, there does not seem a way to represent any sort of fairness. However, the idea that extra data can be offered to be used by a hyper-heuristic is still relevant and could be used in different ways. One example would be the distance measure described in Chapter 5.

6.4 General VRP Domain

The version of HyFlex which this domain is implemented for is the version presented in the previous chapter, which included some new features for HyFlex. One of these new features was a measure of genotypic distance, accessed by a method named *distanceSolutions*. It was demonstrated in the previous chapter that the use of a distance measurement within a hyper-heuristic framework could result in improved solutions. For that domain, the distance measure was calculated with reference to the number of common edges between 2 solutions. The exact formula was $distance = \frac{totalEdges - commonEdges}{totalEdges}$. Due to its successful use in the previous domain, the same measure is used to calculate genotypic distance for this domain.

Another new feature in the previous section was a method, *getFeatureCost(int solutionIndex, int featureIndex)*, which allows access to the values for individual terms of the objective function. This feature allows users to apply their own weights to these terms and hence create their own objective functions within their hyper-heuristic. In theory, this could be useful for investigating fairness. If the distance value for each route was available through this method, then the user could determine how high a workload each route had. In practical terms, though, this is not possible. The method require a *featureIndex*, where an index is given to refer to a solution feature. The index for each solution feature doesn't change. The problem is that the number of routes for different instances will change. It is not possible to say index 4 will return the distance of route 1, index 5 the distance of route 2 etc. because it is not known in advance how many routes there will be and therefore how many solution features must be made available. For this reason, if fairness is to be manipulated through the objective function, it must be done at the level of the problem domain. In this domain, there are 2 solution features available, the total distance travelled and the shortest route. The number of vehicles is not needed here as the number of vehicles for each solution is fixed by the instance file. This reflects industrial applications where a fleet of vehicles will be available of a fixed size.

6.4.1 Additional Features

The following new features all offer opportunity to study fairness using various means. The way this can be done will be explained for each item.

- **Capacity Limits** A capacity limit has been a constraint that has been used in the majority of studied vehicle routing problems. Traditionally, a single value is given which represents the capacity limit for each of the vehicles. This is indeed a constraint present in the real world, where each vehicle will have a finite amount of space. However, this concept can be extended further to better reflect industrial applications where, in reality, a fleet of non-homogeneous vehicles will be available for use. To this end, the domain will allow capacities to be specified for each individual vehicle, all of which can be different or set to be the same in order to represent classical academic problems. The inclusion of this constraint is very useful from a fairness viewpoint because vehicles can be given capacities in such a way as to force the solution to be fair. Consider the example of a small instance of 4 customers, each with a demand of 1 and 2 vehicles. If each of these vehicles is set a capacity of 2, then each vehicle will be forced to service 2 customers and there will be a balanced solution (at least in terms of customers served - not necessarily in distance travelled). This approach works well with the multi-depot feature, which will be described next, along with a thorough example of how the two features may be used together.

- **Multiple Depots** The *Multi-depot Vehicle Routing Problem* is a problem that has received significant attention in the literature. For examples, see [96], [138], [37] and [127]. Again, the motivation for this problem is derived from real-world problems where larger organisations would have multiple distribution centres, each with a certain number of available vehicles. For the domain presented here, each vehicle is assigned to a depot (as specified in the instance file). It is noted in the section above that it can be hard to find the balance between fairness and solution quality. Using the multi-depot feature along with the capacity limits for individual vehicles, it can be measured how solution quality is affected as a solution becomes more or less fair. In Figure 6.1, there are two depots at opposite ends of a co-ordinate space. Each depot has a single vehicle serving it and there are 10 customers dispersed randomly between the two depots. In this example, assume that all customers have a demand of 1. The capacities of the two vehicles can be modified to determine how many customers are serviced by each vehicle. The line in Figure 6.1 shows a situation where the vehicle for depot 1 has a capacity of 4 and thus services 4 customers. The vehicle for depot 2 has a capacity of 6 and services 6 customers. For the fairness testing, all possible capacity values can be considered in turn, starting with values of 0 and 10 for vehicles 1 and 2 respectively, followed by values of 1 and 9 and continuing until values of 10 and 0. For each pair of capacity values, the solution can be optimised using the domain's heuristics. From this, solution quality can be observed at all different levels of fairness and it can be determined whether there is a point at which an increase in fairness causes a vast decrease in solution quality.

- **User Input to Objective Function** The next feature allows fairness to be encourage through the objective function. As was stated in the previous section, the distance values of individual routes are not available to the hyper-heuristic during the search. Therefore, encouraging fairness through the objective function can not be done at the hyper-heuristic layer. The solution proposed here is that a number of 'additions' to the objective function are available to be selected and are to be specified in the instance file.

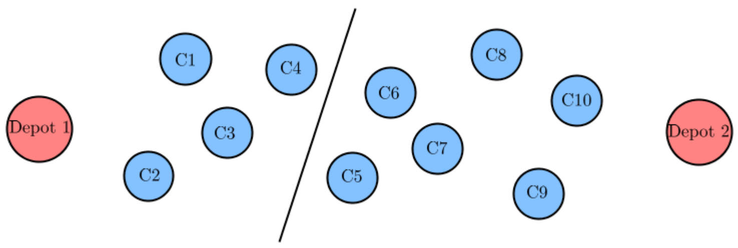


Figure 6.1: Example of a fairness experiment using multiple depots and individual vehicle capacities.

Here, the user has the possibility to define *low* and *high* distance limits and specify penalty amounts to apply upon violation of these limits. By applying penalties to routes that are either overly short or long, solutions will be encouraged to favour routes in between these values. This allows for more flexibility than controlling fairness explicitly through capacity constraints. An example of a line in the instance file to utilise this feature is as follows.

OBJECTIVE : *key low high penalty*

Here, the *key* refers to the choice of objective function addition (see below) and the *low*, *high* and *penalty* values are parameter values to be used in calculation of the objective function. The keys and corresponding additions are as described below:

- *a*: This choice takes three parameter values, *low*, *high* and *penalty*. For each route, the total distance is calculated. Where the distance is either lower than the *low* value or higher than the *high* value, a fixed amount of *penalty* is added to the objective function score. The use of a fixed penalty value discourages violations and can encourage fairness by keeping routes within a certain range.
- *b*: This choice is similar to *a* in that a *low* and *high* parameter values are specified and there is a penalty of *penalty* for any violation. However there is also a variable penalty applied which takes into account the scale of the violation. To calculate this variable penalty, firstly the difference is calculated between the distance and the violated parameter. This value is then multiplied by 100 to give the penalty score to be added. The variable nature accounts for the idea that a route violating

    the range by a small amount will do less damage to fairness than a route violating the values by a large amount.

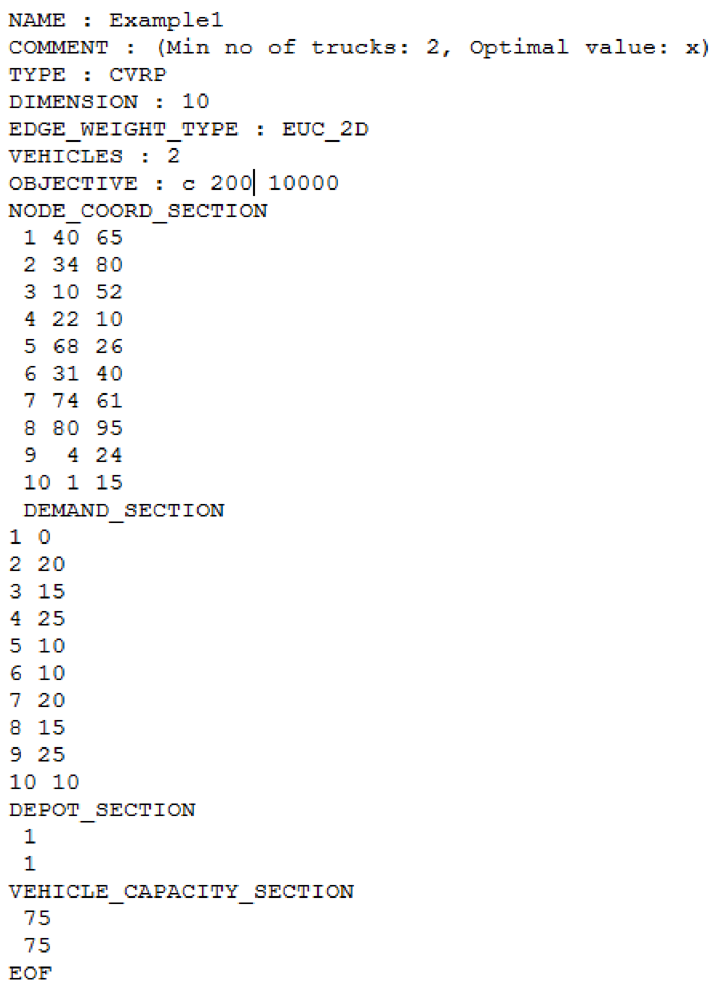
  - *c*: The *c* choice takes 2 parameters, *high* and *penalty*. As with *a*, for each route that has a total distance of greater than the parameter value, a penalty of *penalty* is added to the objective function.
  - *d*: The *d* choice also takes a *high* parameter value. As with *c*, for each route violating the *high* distance value, a fixed penalty of *penalty* is applied. In addition, a variable penalty is applied in the same manner as *b*. That is to say, for each route where a violation occurs, the difference between the distance and the *high* value is calculated. This value is then multiplied by 100 to give the additional penalty value for that route.

- **p-values** One of the most widely-used techniques for encouraging fairness of objective function terms is the *sum of squares (sos)* method. This method exaggerates the levels to which a particular term is unfair. It can be an effective way to force solutions to be fair. However, the effectiveness of the sum of squares method can vary depending on the nature of the problem considered. For some problems, *sos* may be too strong and force fairness at the expense of solution quality. Conversely, for other problems it may be insufficient and not lead to any significant increase in fairness. Again, this returns to the problem of the balance between solution quality and solution fairness. One means of addressing this is proposed by Muklason et al. [111], where the authors suggest using the *sum of powers (sop)* rather than sum of squares. The required value of *p* is likely to be different for different problems and instances. For example, in [111], it is reported that values as high as p=16 are needed (in that case for an examination timetabling problem). The proposal for this domain is to allow a *power (p)* value to be specified by the user through a method *setPValue(double p)*. It may be though that the user wants to change the *p* value throughout the search, for example initially have a low value that finds a high quality solution, before gradually increasing the *p* value to drive the solution toward fairness. In the domain, the *p* value can be changed at any time to allow this.

Representation in Instance Files

This section will describe a new instance format to be used to accommodate the new features and constraints. An example of an instance file in this new format can be seen in Figure 6.2. There are four main data sections of the instance file.

The first is the *NODE_COORD_SECTION* which details x and y co-ordinates for each customer, including the depot(s). A city is identified by an id number. The second section is the *DEMAND_SECTION* which lists the demand value for each of the customers. The third section, the *DEPOT_SECTION*, states which depot is to be used by each vehicle. Finally, the *VEHICLE_CAPACITY_SECTION* states the capacity for each vehicle.



```
NAME : Example1
COMMENT : (Min no of trucks: 2, Optimal value: x)
TYPE : CVRP
DIMENSION : 10
EDGE_WEIGHT_TYPE : EUC_2D
VEHICLES : 2
OBJECTIVE : c 200 10000
NODE_COORD_SECTION
 1 40 65
 2 34 80
 3 10 52
 4 22 10
 5 68 26
 6 31 40
 7 74 61
 8 80 95
 9  4 24
 10 1 15
 DEMAND_SECTION
1 0
2 20
3 15
4 25
5 10
6 10
7 20
8 15
9 25
10 10
DEPOT_SECTION
 1
 1
VEHICLE_CAPACITY_SECTION
 75
 75
EOF
```

Figure 6.2: An example instance file

6.4.2 Construction Heuristic

The constructive heuristic used in this domain could be described as a *Cluster-first, Route-second* method. In this type of method, the customers are firstly grouped into 'clusters' before routes are constructed from these groups. Unlike traditional methods of this sort, this method must be able to adapt to instances with multiple depots. The implementation here works as follows.

The first stage of the algorithm is to cluster the customers. In the instance file, the number of available vehicles is stated, along with the depot to be used for each vehicle. This information is used to create initial empty routes for each vehicle. Following creation of routes, customers are then assigned to routes (but not routed). To do this, the following procedure is applied.

- While the set of un-clustered customers C is not empty, perform the following.
- Select at random a customer, c_1, from C.
- For each route r_1 in the set of routes R, calculate the average distance between c_1 and the customers currently assigned to r_1 (including the depot).
- Select the route which yielded the lowest average distance to c_1 and assign c_1 to that route.
- Remove c_1 from C.

The selection of customers is made randomly so that different solutions will be produced for different seed values, which is important for providing a diverse range of solutions for population-based approaches. When selecting a route, the average distance to the customer is considered so that both the location of the depot and other customers within the route can be considered.

Following this assignment of customers to routes, the routes must then be fully constructed. This is performed with a *Nearest-neighbour* algorithm. Whilst nearest neighbour may not traditionally provide solutions of the highest quality (see [151]), it is more important for a construction heuristic in this domain to provide solutions that have good potential for improvement. This method is designed to give solutions that are of a reasonable quality, without being of so high a quality that algorithms get easily stuck in local optima. This method operates in the following manner.

- For each route r_1 in the set of routes R, perform the following.
- While the set of un-routed customers C_{r1} within route r_1 is not empty, do:
- For each customer c_1 in C_{r1}, calculate the distance between c_1 and the current final customer (not the depot) within the route.
- Insert the customer, c_{best}, which has the lowest distance as calculated in the previous step into the route. The insertion position is as the final customer in the route.

6.4.3 Low-level Heuristics

In the section above describing the additions to the domain, an example was given of how 2 domains could be used, with controlled vehicle capacities to study the affect of enforced fairness on solution quality. However, for that study to be accurate, it is essential that the individual routes are being optimised to a high standard. As stated in the introduction, consistent optimisation results are required so that when fairness is enforced, any changes in the objective function value can be analysed as being a result of the change in fairness, rather than being a result of fluctuations in the performance of the domain and heuristics. Therefore, it must be established whether the low-level heuristics within the domain are strong enough to deliver consistent high quality solutions for single routes. A single route in a VRP can be considered as a *Travelling Salesman Problem (TSP)*. Initially, the set of low-level heuristics from the VRPTW domain was selected for use within this domain. However, preliminary experiments showed performance on benchmark TSP instances to be poor and inconsistent, with an average solution quality of 5-10% worse than the optimal solution. From this, it was clear that the current set of heuristics were insufficient for the required task. To remedy this, 2 actions will be taken. The first is to extend the set of heuristics available in the domain. Specifically, heuristics will be added that have the purpose of reducing single route distance (*2-opt*, *3-opt*, *MoveOne*, *MoveTwo*,*MoveThree*. These additional heuristics will be described below. The second action will be to modify the set of existing heuristics to better cope with the new features of the domain. Again, these changes will be described below.

Mutation Heuristics

The 2 heuristics to be presented here fall in the class of mutation heuristics which seek to modify a solution through one, or a series of, perturbations or neighbourhood moves.

- **Shift Random** The *shift* operator, which is included in the previous VRPTW domain, involves the operation of removing a single customer from a route and re-inserting it into another route. This heuristic will have had little impact on the poor performance in the TSP as it is concerned with intra-route moves, rather than the inter-route moves required for improving TSP instances. Given that previous chapters have shown VRPTW performance to be strong, a success which requires customers to be assigned to, and moved between, routes in an efficient manner, it can be thought that little needs to be change with this heuristic. For that reason, the mechanism for selecting a customer for removal is the same as for the VRPTW *shift*. That is to say, a random customer is chosen from a randomly selected route. The selection is random as this is a mutation heuristic and the objective is to diversify a solution, not improve it directly. Pseudocode for the *shift* method can be seen in Algorithm 4 in chapter 3.

There is a difference, however, in the method used to re-insert a customer into the solution. In the VRPTW *shift* heuristic, the focus of the sub-routine to insert a customer is on finding the best position in the solution for that customer (in terms of 'proximity' to neighbour customers). There are 2 reasons that is not suitable for this domain. Firstly, the new operators to be added to this domain will improve the success of inter-route moves. Therefore, it is not essential for a customer to be inserted into the best position in a route as it can be successfully moved by other operators. Indeed, it may be beneficial to not insert the customer in the 'best' position initially as it may cause a local optimum to be entered. For these reasons, in this new *insertCust* method, the customer is inserted at the end of a randomly selected route (providing it a feasible insertion).

Secondly, the fact that the number of routes are fixed can cause problems as there may be a situation where they are all equally full. Consider a situation where the sub-routine is attempting to insert a customer with a demand of 30 into a solution, which has 5 routes; all of which have a remaining space of 20. Clearly there is sufficient space in the solution for the customer; however no individual route can accommodate it. This is a situation that is not encountered in the VRPTW as an extra route can simply be added. To tackle this situation, a method is proposed whereby, if no route has sufficient space for the customer to be inserted, then the route with the highest amount of available space is selected and the customer from that route with the highest demand is removed. If there still isn't enough space for the original customer, then the process is repeated until there is sufficient capacity. The choice of route and customer are made in order to reduce the amount of operations needed. Once the original customer has been inserted, the removed customers are re-inserted into the solution in the same manner as the original customer. The entire process for the method can be seen in Algorithm 18.

- **Swap** The VRPTW domain contains a mutation heuristic, *Two-opt* which swaps adjacent customers within a single route. To attempt to give the method further scope to mutate solutions, a heuristic *Swap* is proposed for

Algorithm 18 The insertCust(s,c) method which takes as input a solution s and a customer c.

```
procedure INSERTCUST($s, c$)
    if $thereExistsFeasibleRoute(s, c)$ then
        Route $r \leftarrow selectFeasibleRouteAtRandom(s, c)$
        $r \leftarrow insertAtEnd(r, c)$
    else
        CustomerList $C \leftarrow c$
        while $size(C) > 0$ do
            Route $r^1 \leftarrow getRouteWithMostRemainingSpace(s)$
            $r, C \leftarrow removeCustomersFromRoute(r, C)$
            $r \leftarrow addFirstCustToEndOfRoute(r, C[0])$
            $C \leftarrow removeFirstCustFromC(C)$
        end while
    end if
end procedure
```

this domain which swaps two customers within a route. The difference from *Two-opt* is that these customers do not now have to be adjacent. This heuristic involves a single move within a single route. The pseudocode for the *Swap* method can be seen in Algorithm 19. The first stage of the algorithm is to select a route, *r*, at random from the solution. Then, two separate customers, *c^1 and c^2*, are chosen at random from *r*. The positions of these customers are then swapped. The selections are made randomly in order to facilitate generation of a wide range of solutions, which may allow a hyper-heuristic to escape local optima.

Algorithm 19 The *Swap* mutation algorithm takes as input a Solution *s*.

```
procedure SWAPMUTATE($s$)
    Route $r \leftarrow selectRandomRoute(s)$
    Customer $c^1 \leftarrow selectRandomCustomer(r)$
    Customer $c^2 \leftarrow selectRandomCustomer(r)$
    $r \leftarrow swapPositions(c^1, c^2, r)$
end procedure
```

Ruin-recreate Heuristics

A ruin-recreate heuristic will ruin or destroy part of a solution before re-building it. There is a single heuristic for the domain within this category, which will be detailed below.

- **Location-based Radial Ruin** In the VRPTW domain, there are two *ruin-recreate* heuristics which partially destroy a solution before rebuilding it.

One of those heuristics uses the difference in time windows between customers as a measure for determining which customers are to be removed. As the domain proposed here does not include time window constraints, that heuristic can not be used. However, the *Location-based Ruin-recreate* method of the VRPTW domain will be used for this domain. This method uses the euclidean distance between customers to determine which customers to remove. The 'ruin' element of the heuristic is the same as that used for the VRPTW domain as it is compatible with the new features of this domain. The pseudocode for this can be seen in Algorithm 7 in Chapter 3. The re-insertion, however, will now use the *insertCust* method described above (Algorithm 18, to account for the fixed numbers of routes.

Local Search Heuristics

The category of local search or hill-climbing heuristics contains the most heuristics of all classes within this domain. These heuristics make one, or a series of small moves or perturbations to a solution and only accept improving solutions.

- **Shift Random Local Search** This heuristic is the same as the *Shift Random* method described in the mutation heuristic category (section 6.4.3). The only difference in this version is that, at each iteration of the loop, the new solution resulting from the move is only accepted if it yields an improved objective function value.

- **2-opt** Through the inclusion of the *2-opt* algorithm of [98], it is thought that the quality of individual routes can be greatly improved. The algorithm was first proposed for the *Travelling Salesman Problem(TSP)*, for which strong results can be gained. Given that this is the case when *2-opt* is used in isolation, when combined with other local search heuristics, hyper-heuristics should be able to achieve high quality routes. The general concept is to remove 2 edges from a solution and replace them with 2 new edges, with the goal of reducing the distance of the cycle/route. an example of a *2-opt* move can be seen in 6.3. Algorithm 20 shows the entire process used by the 2-opt implementation of this domain. It differs from the Lin [98] implementation in that the Lin version includes some reductions of the search. These reductions are not currently included in the 2-opt of this domain; however, they would be useful for future implementations in order to improve efficiency which would be crucial for very large problems.

  The aim of this heuristic is to increase the quality of a single route by the highest level possible. To this end, the *2-opt* implemented here considers all

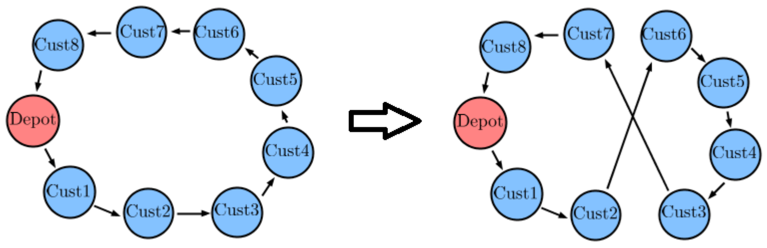


Figure 6.3: Example of 2-opt. Two edges are removed, the edge between customers 2 and 3, and the edge between customers 6 and 7. These edges are replaced by edges between customers 2 and 6, and 3 and 7. The orientation between 3 and 6 is reversed.

possible pairs of edges in a solution. While this may cause issues of processing time for very large problems, it will be of sufficient speed for the vast majority of real-world instances, where individual routes will not be of a great enough size to cause issues. Therefore, it seems a worthwhile trade-off.

Algorithm 20 The *2-opt* local search algorithm takes as input a Solution s.

```
procedure TWO-OPT($s$)
    Route $r \leftarrow selectRandomRoute(s)$
    for all Edge $e^1(c^1, c^2) - in - r$ do        ▷ $c^1$ and $c^2$ are consecutive customers
        for all Edge $e^2(c^3, c^4) - in - r - after - e^1$ do        ▷ $c^3$ and $c^4$ are consecutive customers
            if $newEdgesWillYieldImprovement(e^1, e^2)$ then
                $r \leftarrow removeEdges(e^1, e^2, r)$
                $r \leftarrow insertNewEdges(c^1, c^2, c^3, c^4, r)$
            end if
        end for
    end for
end procedure
```

- **3-opt** The *3-opt* heuristic is in the same class of solution improvement methods, *k-opt* heuristics [98], as the *2-opt* operator described above. In the Lin paper [98], it is shown to be even more powerful than *2-opt* for improving TSPs (individual routes) and therefore is a logical inclusion in the domain. The difference from 2-opt is that 3-opt removes and inserts 3 edges, as opposed to the 2 of 2-opt. There are 2 possible *3-opt* moves for each set of removed edges. These can be seen in Figures 6.4 and 6.5. Again, the 3-opt algorithm for this domain differs only from that of Lin through the absence of a problem reduction stage within this algorithm. The pseudocode for this version of 3-opt can be seen in Algorithm 21.

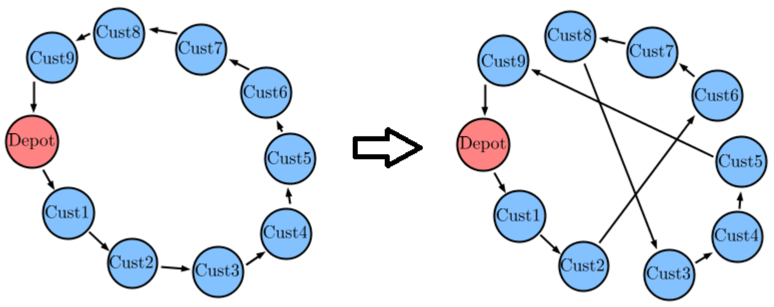


Figure 6.4: First example of 3-opt. Three edges are removed, {2,3}, {5,6} and {8,9}. These edges are replaced by edges {2,6}, {8,3} and {5,9}.

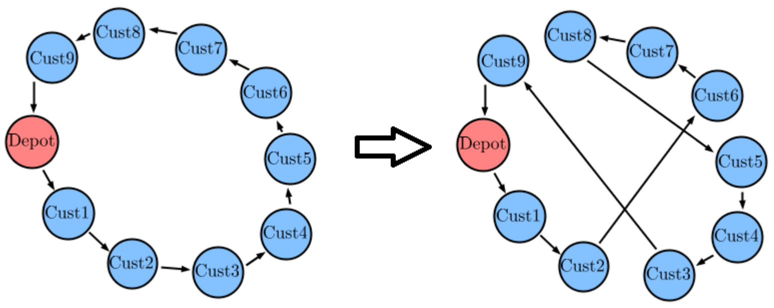


Figure 6.5: Second example of 3-opt. Three edges are removed, {2,3}, {5,6} and {8,9}. These edges are replaced by edges {2,6}, {8,5} and {3,9}. The orientation between 3 and 5 is reversed.

As with *2-opt*, this *3-opt* implementation considers all possible combinations of edges to be removed within a route. With *3-opt*, this is even more inefficient; however still not at a level that would cause problems for the testing of fairness or for solving real-world problems. One implementation detail that helps reduce computation time is that, for each possible move, a calculation is made in advance as to whether the move will result in an improved solution. Therefore, if the move would not improve the solution, it does not have to be made and time is saved.

- **Interchange** In the VRPTW domain, an *interchange* heuristic is described which swaps two customers from separate routes. The aim of the heuristic is to reduce total distance by relocating customers that may not be in the optimal route. The same heuristic as was described in Algorithm 6 in Chapter 3 is included in this domain. However, due to different features of the 2 domains, the following changes must be made.
  - The selection of the first customer to be swapped is made in the VRPTW version of interchange through use of a measure of that customer's distance and time window proximity to other customers of the same route. However, the lack of a time window constraint in this domain means

Algorithm 21 The *3-opt* local search algorithm takes as input a Solution s.

```
procedure THREE-OPT($s$)
    Route $r \leftarrow selectRandomRoute(s)$
    for all Edge $e^1(c^1, c^2) - in - r$ do        ▷ $c^1$ and $c^2$ are consecutive customers
        for all Edge $e^2(c^3, c^4) - in - r - after - e^1$ do        ▷ $c^3$ and $c^4$ are consecutive customers
            for all Edge $e^3(c^5, c^6) - in - r - after - e^2$ do        ▷ $c^5$ and $c^6$ are consecutive customers
                if $newEdgesWillYieldImprovement(e^1, e^2, e^3)$ then
                    $r \leftarrow removeEdges(e^1, e^2, e^3, r)$
                    $r \leftarrow insertNewEdges(c^1, c^2, c^3, c^4, c^5, c^6, r)$
                end if
            end for
        end for
    end for
end procedure
```

that this measure can no longer be used. Instead, only the euclidean distance from its neighbouring customers shall be considered. The distance is used as a customer which is far from its neighbouring customers will drive the total distance value higher, which is undesirable in terms of the objective function. The formula to be used to determine which customer will be selected is:

$score = (dist(c_{i-1},c_i + dist(c_i,c_{i+1})$*randomNumber

In the formula, the distance is calculated between the customer and the customers both preceding and following it in the route. The random number is there to ensure that the same customer doesn't get selected at every iteration of the algorithm, which could lead the algorithm to being repeatedly unsuccessful.

– The selection of a second route from which a second customer will be selected is the same in both domains. However, the selection of the second customer itself is different. For the VRPTW domain, a measure of distance and time proximity to the first customer is used. For this domain, that again isn't possible. Rather than simply using a distance proximity measure to select a customer, each possible customer from the second route is considered with the new objective function calculated for if the swap is made. This approach is used to attempt to produce the highest quality solution.

- **Swap LS** This heuristic performs the same operation as the *Swap* heuristic in the mutation category (see section 6.4.3 and Algorithm 19). There are two difference in the local search version of the algorithm, which are described below.

- In the mutation version of the algorithm, a swap move is performed only once. In the local search version, the move is performed multiple times, with the number of times to be performed being determined with reference to the *depthOfSearch* parameter (see Chapter 3 for an explanation of this parameter). The formula used to calculate the number of times the move is to be performed is as follows;

  $timesToPerform = numberOfRoutes * depthOfSearch$

  The parameter is multiplied by the number of routes so that the scale of intensification can adjust to the size of the problem.
- The second difference concerns the acceptance of the new solution following a move. In the mutation version of swap, the resulting solution is always accepted, regardless of whether the solution is improved or not. For the local search version of the method, at each iteration the new solution is only accepted if it has an improved objective function value.

• **MoveOne** The *MoveOne* heuristic is included in the domain with the objective of improving performance on single routes. It achieves this by removing a single customer and then inserting it into a different position, but within the same route. This heuristic provides a different way to manipulate a route to the heuristics which were previously included and so has the potential to improve performance. The pseudocode for this algorithm can be seen in 22. As with some of the other local search heuristics, MoveOne performs a number of moves, determined as the value of the *depthOfSearch* parameter multiplied by the number of routes, so that the number of applications is proportional to the size of the problem. It is important that many possible moves are considered, so that the method doesn't get stuck at a local optimum. For that reason, at each iteration, a route r is selected at random from the solution, s. Similarly, from r, a customer c is selected, also at random, and removed from r. Following removal, the aim is to improve the solution. To do this, the algorithm enters a *First-Improvement* stage. In first-improvement, a series of moves are independently performed until an improvement in objective function is found, at which point the algorithm stops. First improvement is chosen over the *best-improvement* method in order to both save computational time and avoid the local optima that may arise from a best improvement approach. In the context of this method, the customer c is sequentially inserted at each possible insertion point within r (starting with insertion after the vehicle leaves the depot and finishing with insertion as the final stop before the depot) until an insertion yields an improvement in objective function value, at which point the first-improvement stage stops and the solution is accepted as the current solution. The algorithm then continues until it has been performed the required number of times.

Algorithm 22 The *MoveOne* local search algorithm takes as input a Solution s.

```
procedure MoveOne($s$)
    $timesToPerform \leftarrow (numberOfRoutes * depthOfSearch)$
    for $i \leftarrow 0, timesToPerform$ do
        Route $r \leftarrow selectRandomRoute(s)$
        Route $r' \leftarrow r$
        Customer $c \leftarrow selectRandomCustomer(r')$
        $r' \leftarrow removeCustFromRoute(r', c)$
        for all Customer $c'$ in $r'$ do
            $r' \leftarrow insertCustBeforeCust(c, c', r')$    ▷ Inserts customer c before c'
in route r'
            if $objFunc(r') < objFunc(r)$ then
                $r \leftarrow r'$
                break
            else
                $r' \leftarrow removeCustFromRoute(r', c)$    ▷ Undo move
            end if
        end for
    end for
end procedure
```

- **MoveTwo** *MoveTwo* follows the same structure and set of operations as MoveOne. It provides a new means of improving the quality of single routes. It differs from *MoveOne* in that it is a pair of customers which are removed and re-inserted, rather than just a single customer. The pair of customers are consecutive customers, both in the original route and the new route. The algorithm for MoveOne can be seen in Algorithm 22.

- **MoveThree** *MoveThree* extends the methods of MoveOne and MoveTwo by removing and re-inserting three consecutive customers at a time to provide a further way of manipulating a single route. The implementation of the method is otherwise the same as for MoveOne (Algorithm 22).

6.5 Hyper-heuristic Approach for Minimising Route Cost

The previous section describes how a set of low-level heuristics have been chosen with the goal of improving performance on optimising single routes (in other words, the TSP). The motivation behind that is the need for high quality routes in order that testing of solution fairness can be interpreted correctly. This section will present a hyper-heuristic that successfully utilises the set of low-level heuristics to generate high quality single routes. To demonstrate whether this is the case, testing will be performed on benchmark instances of the *Travelling Salesman Problem (TSP).*

As the aim of the hyper-heuristic is to demonstrate the performance (or at least potential) of the low-level heuristics, the algorithm itself will be kept simple so that any success can be attributed mainly to the set of low-level heuristics. It will be a population-based algorithm as that proved successful for previous routing problems (see Chapter 5). It will include a mutation stage to diversify solutions and escape local minima and a local search stage to intensify the search and improve solution quality. Full pseudocode for the hyper-heuristic can be seen in Algorithms 23 and 24. Each stage of the algorithm will also be described below.

- **Setup and Variables** When considering what size a population must be, a balance must often be found between the improved solution quality that results from a large, diverse population and the increase in time taken to improve this population of solution. From preliminary testing, a population size of 4 proved to provide the best balance within this hyper-heuristic. As well as ensuring a level of diversity within solutions, it is also important to determine when a solution should be intensified and when it should be diversified. To represent this notion, a new method of adaptation of the *intensityOfMutation* HyFlex parameter will be proposed. In this algorithm, each population member has a separate value for the *intensityOfMutation* parameter, a parameter which controls the scale to which a solution is mutated by a mutation heuristic. The variable $intOfMut_i$ is used to refer to the *intensityOfMutation* value for the solution with index i. When a solution is still being improved, changes in a solution resulting from the application of a mutation heuristic should be relatively small. To this end, an initial *intOfMut* value of 0.4 is given for all solutions, which will provide enough of a change to escape small local optima but without changing the solution to such a degree that previous improvements are rendered null. For each solution, there is also a variable tracking how many iterations it has been since an improvement in objective function value. This variable is $itsSinceImpr_i$, again with the i referring to the solution index. The final variable, which can be seen on line 9 of the pseudocode, is $lsScore_i$, which gives the 'score' for the local search heuristic with index i. The score is used to select heuristics in the local search stage of the algorithm. Initially, all heuristics must be given a high enough score so that early improvements for certain heuristics do not overpower other heuristics. Preliminary testing proved that a value of 100 was suitable for these purposes. Following this setup, a loop is entered where the following stages are performed for each solution in turn.

- **Adjustment of Mutation Strength** As mentioned in the previous paragraph, the intensity of mutation is to be adapted for each solution, depending on the state of the search. Lines 13-20 of Algorithm 23 show how this is manifested in the hyper-heuristic. Line 13 checks whether there have been 2 or more iteration since an improvement in objective function value has been found. 2 is an arbitrary value chosen due to the fact that the strength of the local search stage can mean that a solution is stuck in a local optima after even a single iteration. If this is the case, it is then checked whether the $intOfMut_i$ variable is at its maximum allowable value of 1 (as defined by HyFlex). If it is, then the implication is that several varying levels of mutations have been performed without an improvement in solution quality. In order to escape the local optimum it would seem that the solution is in, the solution is re-initialised and the $intOfMut_i$ value is set back to 0.4 (lines 15 and 16). If the $intOfMut_i$ variable is not at its maximum level, then it is increased by 0.1, again an arbitrary value chosen to ensure that several mutation heuristics can be applied before the solution will be re-initialised (line 18).

- **Mutation Stage** As stated at the start of this section, this hyper-heuristic is deliberately simple in its design. Another consideration for the mutation stage is that there are only 3 mutation heuristics (including the ruin-recreate method). Therefore a sophisticated selection mechanism would be rendered less useful. For these reasons, the selection of a mutation heuristic is made at random and a single application is made to the solution.

- **Local Search Stage** The local search stage of the algorithm can be seen as pseudocode in Algorithm 24. Again, the aim is to maintain simplicity as far as possible. However, the large number of heuristics in this category (8) means that selecting heuristics at random may be inefficient. To handle this problem, a re-inforcement learning method will be used for selection of local search heuristics. The selection method will be simple enough to not add unwanted complexity to the algorithm whilst allowing for a level of heuristic intelligence. At each iteration of this method, a local search heuristic is repeatedly selected and then applied to the current solution. Again, a tracker is used to record how long it has been since an improvement in objective function value. It is important to identify how many iterations should be performed without success before the stage is exited so that a mutation heuristic can be applied. If the value is too low, potential improvements will be missed. If too high, computational time may be wasted. Testing of a small number of various value indicated that a strong, if not necessarily optimal, number of iterations at which to stop was 30. At each repetition, a heuristic is selected by the *Roulette Wheel Selection Mechanism* (see Algorithm 13 in Chapter 4) which uses the heuristics' $lsScore_i$ scores. If the application of this selected heuristic results in an improvement in objective function value, a value of 1 is added to its 'score' (the $lsScore_i$ variable). The 1 is selected as a small value that will reward the operator without allowing it to overpower other operators from a small number of successful applications.

6.5.1 Experimental Setup

The hyper-heuristic described in the previous section is to be tested on the TSP to demonstrate whether the set of low-level heuristics are suitable for use in experiments into fairness. The experimental setup will be described below. The memetic algorithm described above shall be referred to as *MemAlgReinforce* or MAR, due to the element of reinforcement learning in the local search stage of the algorithm.

Instances

The instances to be used here can be found in TSPLIB [136], a problem library which has been widely studied. As well as being a well-known and well-studied dataset, optimal solutions are available for all instances, allowing for meaningful comparisons. For the results, the instances will be considered in 2 sets; those with less than 200 customers and those with 200 customers or more. In the former set, there are 22 instances and for the latter category 14 instances. The largest instance has 575 customers, a number of customers far greater than would be present in a single route of a real-world routing problem. Thus, if successful performance can

Algorithm 23 The *Memetic Algorithm with Reinforcement Learning* hyper-heuristic

```
procedure MEMETIC ALGORITHM
    $popSize \leftarrow 4$                                   ▷ Population size
    for $i \leftarrow 0, popSize$ do
        Solution $s_i \leftarrow initialiseSolution(i)$
        $intOfMut_i \leftarrow 0.4$          ▷ intensityOfMutation value for solution $i$
        $itsSinceImpr_i \leftarrow 0$        ▷ Iterations since improvement for solution $i$
    end for
    for $i \leftarrow 0, noOfLSHeuristics$ do  ▷ For the no. of local search heuristics
        $lsScore_i \leftarrow 100$                    ▷ The 'score' for LS heuristic $i$
    end for
    while $timeHasNotExpired()$ do
        for $i \leftarrow 0, popSize$ do
            if $itsSinceImpr_i \geq 2$ then
                if $intOfMut_i == 1$ then
                    $s_i \leftarrow initialiseSolution(i)$         ▷ Re-initialise solution
                    $intOfMut_i \leftarrow 0.4$                     ▷ Re-set $intOfMut_i$
                else
                    $intOfMut_i \leftarrow (intOfMut_i + 0.1)$      ▷ Increase $intOfMut_i$
                end if
            end if
            Solution $tempSol \leftarrow s_i$
            $mutHIndex \leftarrow selectRandomMutationHeuristic$
            $tempSol \leftarrow applyHeuristic(mutHIndex, tempSol)$
            $tempSol \leftarrow localSearchStage(tempSol, lsScore)$   ▷ See Algorithm 24
            if $objFunc(tempSol) < objFunc(s_i)$ then
                $s_i \leftarrow tempSol$
                $itsSinceImpr_i \leftarrow 0$
            else
                $itsSinceImpr_i \leftarrow (itsSinceImpr_i + 1)$
            end if
        end for
    end while
end procedure
```

Algorithm 24 The *local Search Stage* sub-routine for the Memetic Algorithm with Reinforcement Learning.

```
 1: procedure LOCAL SEARCH STAGE($s, lsScore$)
 2:     $itsSinceImpr \leftarrow 0$
 3:     $itsLimit \leftarrow 30$
 4:     while $itsSinceImpr < itsLimit$ do
 5:         Solution $s' \leftarrow s$
 6:         $ind \leftarrow rouletteSelection(lsScore)$   ▷ $ind$ is the index of the heuristic.
    See Algorithm 13 in Chapter 4 for $rouletteSelection$ method.
 7:         $s' \leftarrow applyHeuristic(ind, s')$
 8:         if $objFunc(s') < objFunc(s)$ then
 9:             $s \leftarrow s'$
10:             $itsSinceImpr \leftarrow 0$
11:             $lsScore_ind \leftarrow (lsScore_ind + 1)$
12:         else
13:             $itsSinceImpr \leftarrow (itsSinceImpr + 1)$
14:         end if
15:     end while
16: end procedure
```

be shown on a large instance such as this, then it will be implied that the domain is suitable to accurately investigate fairness.

Test Details

For each of the instances in the first set of problems (those of 200 customers and less), 10 runs will be performed of 10 CPU minutes each. For the larger instance, there will be 10 runs each of 20 CPU minutes length. A machine with a 2.27GHz Intel(R) Core(TM) i3 CPU and 4GB RAM will be used to run the tests.

6.5.2 Results

As mentioned previously these instances have been split into 2 sets, with instances of less than 200 customers in one set and instances with 200 and more customers in a separate set. Each instance set will have a single table representing the results over those instances. The table gives an instance index as well as a name. Within the instance name is a number, which represents how many customers are in that instance. The instances are ordered within the table in increasing order of the number of customers. Both the median and best-found results are presented for the *Memetic Algorithm with Reinforcement learning (MAR)* algorithm. The known optimal results are also presented for comparison purposed. To aid comprehension of the results, the percentage difference between the best result found by the MAR method and the optimal solution is presented. It is important to note when considering the results that the distance measurement used for the optimal solution results rounds the distance for each edge. This is in contrast to the distance calculation within the domain, which is pure Euclidean distance. Thus, there are some small anomalies; for instance some results that are lower than the optimal result. The results for the smaller instances can be found in Table 6.1 and the results for the larger instances can be found in Table 6.2.

Instance	Name	MAR-median	MAR-min	Optimal	% Diff.
0	eil51	428.87	428.87	426	0.674
1	berlin52	7544.37	7544.37	7542	0.031
2	st70	677.11	677.11	675	0.313
3	eil76	544.37	544.37	538	1.184
4	pr76	108159.44	108159.44	108159	0
5	rat99	1219.24	1219.24	1211	0.68
6	kroA100	21285.44	21285.44	21282	0.016
7	kroB100	22139.07	22139.07	22141	-0.009
8	kroC100	20750.76	20750.76	20749	0.009
9	kroD100	21294.29	21294.29	21294	0.001
10	kroE100	22068.76	22068.76	22068	0.003
11	eil101	641.23	641.21	629	1.941
12	lin105	14383	14383	14379	0.028
13	pr107	44301.68	44301.68	44303	-0.003
14	pr124	59030.74	59030.74	59030	0.001
15	bier127	118370.83	118293.52	118282	0.01
16	pr136	96875.82	96770.92	96772	-0.001
17	pr144	58535.22	58535.22	58537	0.003
18	kroA150	26610.80	26524.86	26524	0.003
19	kroB150	26218.16	26138.70	26130	0.033
20	pr152	73683.64	73683.64	73682	0.002
21	rat195	2360.11	2353.66	2323	1.32

Table 6.1: A table showing the results for the smaller TSP instances.

By considering the results for the smaller instances (Table 6.1), it is evident that the heuristics and domain are very capable of solving TSP problems of that size. The percentage difference from the optimal solution is very low for all of the instances, with the worst difference being 1.32% and only 2 cases being present where the difference is greater than 1%. By looking at the results for the larger instances (Table 6.2), it can be seen that the performance of the algorithms holds up well. Although the highest percentage difference rises to 3.899%, the average difference is only 1.022%, a level that would be considered as acceptable for real-world applications. One point that should be noted is that for the largest of the instances tested, the performance does seem to deteriorate slightly. It is most likely that this is due to the greater amount of time needed to perform some of the low-

Instance	Name	MAR-median	MAR-min	Optimal	% Diff.
0	kroA200	29478.44	29439.50	29368	0.243
1	kroB200	29827.50	29440.82	29437	0.013
2	ts225	126645.93	126645.93	126643	0.0023
3	pr226	80679.21	80370.26	80369	0.0016
4	gil262	2429.70	2405.21	2378	1.144
5	pr264	49274.18	49135	49135	0
6	a280	2655.15	2622.50	2579	1.687
7	pr299	49120.95	48599.53	48191	0.848
8	lin318	42975.30	42484.27	42029	1.083
9	pr439	111842.73	107924.53	107217	0.66
10	pcb442	52629.85	52139.91	50778	2.682
11	rat575	7127.02	7037.09	6773	3.899

Table 6.2: A table showing the results for the larger TSP instances.

level heuristic operations as the size of the problem grows larger. Specifically, the 2-opt and 3-opt methods could be considered slightly in-efficient. This could be improved in both cases by implementing the problem reductions proposed by Lin [98] for these methods. These problem reductions restrict moves to only those that have the possibility of imporving a solution. However, for the primary motivation behind this domain, that of investigating and encouraging fairness between routes, the performance is sufficient to suggest that the domain can provide a strong base for such testing.

6.6 Conclusion

This chapter has looked at the design decisions that must be considered when creating problem domains for real-world routing problems. Particular emphasis has been placed on the issue of 'fairness' between solutions. An implementation of this concept has been provided for a Vehicle Routing Problem domain, along with analysis of how fairness could be represented for the other HyFlex domains. These fairness definitions across multiple domains are significant as they would allow a hyper-heuristics to perform fairness experiments in a cross-domain context. The work in this chapter has shown some of the complexities behind design of real-world components within problem domains. For fairness within VRP, several tools had to be added to the domain. These could then be used as needed by a hyper-heuristic. These varied from enforcing fairness (through vehicle capacities) to encouraging fairness (using p-value to alter objective function terms).

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis has explored the concept that, through informed design of problem domains and with the provision of sufficient tools and data, a hyper-heuristic can be made more powerful and flexible. In order to successfully address this idea, a number of angles have been considered. One of these has been the definition of what a problem domain for a hyper-heuristic actually is, as it will inevitably have different requirements to a domain designed with other algorithms in mind. Following on from this, analysis and discussion of the various domain components has been given. This analysis has covered how they can be designed to enable a hyper-heuristic to operate efficiently. The design of a domain also includes decisions about what information should be available to a hyper-heuristic. This thesis has explored how a hyper-heuristic can use extra information to more easily navigate the search space and obtain improved results. Part of this exploration has included analysis of the hyper-heuristic element of optimisation, as well as just the problem domain. The work has shown there is a important relationship between problem domain and hyper-heuristic. The domain must provide sufficient information and tools but the algorithm must be designed to make use of that information. The thesis has also considered how domain design can differ for real-world applications. One particular example of fairness has been examined. The analysis shows that careful consideration must be given when designing such a domain to ensure that, not only is the concept of fairness implemented, but that the tools are available for a hyper-heuristic to achieve improved fairness. The contributions of each chapter will now be analysed in more detail.

7.1.1 Problem Domain Definition and HyFlex VRPTW Domain

A substantial contribution of Chapter 3 is the definition of a problem domain for hyper-heuristics. With components divided into 2 categories of *Base Representation* and *Domain Tools*, the definition describes the elements of a problem domain that are needed for a wide variety of hyper-heuristics to be able to operate on it. In the chapter, each of the elements of the domain have been analysed and the design decisions behind them discussed. This discussion has occurred both in a theoretical manner and also with a practical example. The practical example was a new domain for the Vehicle Routing Problem with Time Windows for the HyFlex framework. By presenting this domain, we were able to give concrete examples of some of the design decisions faced. One of these decisions is the choice of low-level heuristics to be included in the domain. A key contribution of this domain is the combination of many of the most used and best performing low-level heuristics from the literature. In addition, 2 new heuristics were proposed in the crossover heuristic category. One of these heuristics selects routes randomly from the 2 parent solutions. The other of these new heuristics attempts to select 'long' routes from the 2 parent solutions, working from the motivation that a long route is beneficial as, by including more customers in a single route, it may be possible to reduce the number of routes required and also the distance travelled.

The analysis of design of problem domains raised some interesting points. Whereas the design of a domain for a meta-heuristic or more exact approach might concentrate almost exclusively on permitting improved performance on a single problem and small set of instances, a domain for a hyper-heuristic must be designed with flexibility in mind. The design decisions discussed in Chapter 3 reflect this. The low-level heuristics are designed to take a short amount of time to run on each call. This ensures that multiple calls can be made by a hyper-heuristic and more information can be gathered. Data as a driver of hyper-heuristic performance is the driving force of this thesis and is a crucial element of domain design. The choice of heuristics is also important, with a broad range of 'type' of heuristic, allowing many different hyper-heuristics to operate on the domain.

7.1.2 Information as a Driver of Hyper-heuristic Performance

One of the key contributions from Chapter 3 had been the need for a domain to provide a hyper-heuristic with the right tools and data. Chapter 4 seeked to explore this concept further. To achieve this, 2 hyper-heuristics which take inspiration from different research areas were proposed. Both algorithms followed an *Iterative Local Search* framework. The first is the *Choice Function* [41] from the field of hyper-heuristic research. The choice function considers 3 measures of a heuristic's success; its ability to improve objective function, its performance when it is applied immediately following another heuristic and the time taken for it to complete its application. The second approach is *Adaptive Operator Selection (AOS)* from the research area of evolutionary algorithms. In that research, the selection mechanism is used to select which heuristic should be used to mutate a population member. The specific version of AOS used in this chapter (*Extreme-value Adaptive Operator Selection* [58]) considers the past successes of heuristics in the search and favours heuristics which have shown the ability to make single, large jumps in terms of objective function improvement.

These 2 hyper-heuristics were compared to an ILS algorithm which had a random mutation heuristic selection mechanism. They were both able to produce consistently better performances (in terms of objective function score). The main difference from the base algorithm was the use of a mutation heuristic selection mechanism based on past performance data. These results provided a strong indication of the value that can be added by providing more data to a hyper-heuristic. This theory was further tested with the proposal for an adaptive mechanism for the local search stage of the ILS method, as well as for the mutation stage. Using the insight gained from the previous comparison of selection mechanisms, a simple method is used which orders the local search heuristic using their average improvement to objective function value. At each iteration, the heuristics are repeatedly applied in the determined order until no improvement is found. This new hyper-heuristic was tested on the new VRPTW domain proposed in Chapter 3 and yielded improved objective function scores compared to the previous version. Again, this strengthens the conclusion that the heuristic performance data offered to the hyper-heuristic is crucial to its performance when used well.

The other interesting contribution to come out of chapter 4 was the analysis on the role of parameter numbers in hyper-heuristic performance. For the initial testing and comparison of the two hyper-heuristics with the adaptive mutation heuristic selection mechanisms, there was found to be a measurable performance difference between the two. The AOS algorithm provided superior (in objective function values) as well as more consistent results than the *Choice Function* hyper-heuristic. Both algorithms had the same underlying structure. The main difference between them was in the nature of the mutation heuristic selection mechanism. The AOS method only required a single parameter to be tuned for the mutation stage. The *Choice Function* had 3 parameters. The analysis in Chapter 4 postured that the lower number of parameters in the AOS method allowed it to have more flexibility in the face of changing problems and instances. The logic behind this was that parameters will inevitably have optimal settings for different problems. It will be very unlikely that these values are the same for all problems. Hence, it could be argued that the greater the number of parameters, the greater the amount of tuning that is needed when moving between problems. This is an interesting contribution which would merit further investigation. Suggestions as to how this could be done are given later in this chapter.

7.1.3 New Domain Tools

Chapter 5 continued the work of Chapter 4, in exploring further how extra data and tools could be made available to a hyper-heuristic in such a way as to allow that hyper-heuristic to better navigate the search space and achieve improved solutions. In order to do this, Chapter 5 proposes several additions to the HyFlex framework, all intended to enhance the framework and provide more options to hyper-heuristics. One of these improvements is access to solution 'features'; qualities of a solution that can be used to judge quality or to drive the search a particular way. Chapter 5 describes how access to these solution features can be used to generate user-defined objective functions. An example is given of the VRPTW, where an algorithm may want to first optimise the number of routes before then optimising the distance. Further improvements include the possibility to save and load solutions, which means strong solutions can be used as seed members of a population in an evolutionary algorithm. A further feature is a method which allows external instances to be imported into a domain. Through this, users can import their own instances and increase the scope of testing of algorithms. This element of the work presented in Chapter 5 provides an important contribution in understanding of how domain tools can be used by hyper-heuristics.

One of the most significant additions is that of a solution 'distance' measure which gives a value representing how 'different' 2 solutions are. There are a number of applications of this feature, for example in hyper-heuristics with genetic algorithm elements where population diversity is desirable. To demonstrate the utility of this feature in full, definitions of distance for all HyFlex domains were given, with options considered. In addition, a distance measure was implemented for the VRPTW domain that combined quick execution time when calculating the distance value with the need to provide an accurate measure. These definitions given in the context of distance for use by hyper-heuristics represents a valuable contribution with the potential for much further work.

In order to directly demonstrate how the use of a distance measure can benefit a hyper-heuristic, an example application is considered and implemented in Chapter 5. This is the proposal to assess quality of heuristics by the level to which they 'change' a solution as well as whether that solution is improved. Specifically, a method called *Compass* [104] is used to find a balance between solution modification and solution quality. This method is used in the context of a memetic algorithm and is used to select both the mutation heuristics and the local search heuristics. The distance measure is compared to a simple reinforcement learning technique which is inspired by the above conclusions drawn regarding parameter numbers. Both approaches are tested on the VRPTW domain and are shown to have similar levels of median performance. However, the best of the results using the distance measure method are often stronger than the best results for the other method. It is very interesting that the results show a distance measure can be used by a hyper-heuristic to provide improved results. However, it is clear that further work is required to determine more applications of the tool. Finally, the results also showed that the reinforcement learning technique outperforms all competitors from the *CHeSC* competition for the VRPTW domain, indicating once again the strength of simple techniques with few parameters.

7.1.4 A New VRP Domain with Real-world Features

Chapter 6 considered the issue of domain design and provision of tools for hyper-heuristics from the perspective of real-world routing problems. A new HyFlex domain with features designed to represent issues that may be encountered in industrial routing applications. The specific issue of fairness between routes is considered, with several features being provided to allow manipulation and analysis of solution fairness. One important contribution of this chapter is a definition of fairness for each of the HyFlex problem domains. These definitions show how the concept of fairness is applicable to many different problems. In addition, practical implementations are given for the new routing domain. All of the new features in the domain are described, with explanations given as to how they can be used to investigate fairness. One new feature is individual capacities for each vehicle in a solution. Using this feature, fairness can be forced by manipulating vehicle capacities to ensure each vehicle has a certain number of customers. In this manner, it can be investigated how enforced fairness can affect the potential best objective function value. Another feature is proposed which allows fairness to be encouraged through the objective function. Through this feature, users can place soft limits on route distance which are penalised by both fixed and variable penalties. A further method is included to encourage fairness through the objective function, a method which calculate the sum of ${dist_r}^p$ for all routes r within the solution. This is a more flexible approach than the *sum of squares* method. By providing this analysis and explanations of how these features can be used to investigate fairness, Chapter 6 makes a strong contribution to knowledge in the area and provides a solid base for future work.

As mentioned, the new features have the potential to facilitate investigation into the link between fairness and solution quality. In chapter 6, it is explained that the quality of solutions generated from the domain must be consistent; otherwise it is hard to draw conclusions about whether any change in performance is down to fairness or just the fluctuating performance of the domain's heuristics. With this in mind, Chapter 6 mentions that the original set of heuristics for the VRPTW domain do not provide strong, consistent solutions for single routes. To remedy this, a new set of heuristics are described and implemented in this new domain, with a focus on heuristics that are able to make inter-route improvements. Much stronger results are demonstrated on single routes using these heuristics. This once again shows how a strong combination of low-level heuristics can be vital to achieving consistent results.

7.1.5 Summary

The proposition that this thesis has put forward is that, for a problem domain to be of use to a hyper-heuristic, it must offer enough tools and data for the hyper-heuristic to effectively navigate a search space. A number of contributions have been presented over the chapters which show that a hyper-heuristic's potential to achieve competitive results can increase as it is offered more data to interpret. The work done demonstrates how design of problem domains (and particularly Vehicle Routing Problem domains) for hyper-heuristics requires different factors to be considered than when designing for other algorithms. This thesis has analysed these decisions and has provided a firm base for future design of domains for hyper-heuristics.

7.2 Future Work

This section will describe possible work that can be undertaken that builds on the contributions of this thesis.

- The distance measure introduced in Chapter 5 could be extended by HyFlex. Firstly, by other domains implementing it (as well as the other changes) so it can be tested on multiple problems. Secondly, multiple distance measure could be available for each problem domain. This would increase the amount of data available to a hyper-heuristic and increase the potential for manipulation of the search space. On a similar note, a number of different distance measures for the VRPTW should be tested within the HyFlex framework. Currently, only one has been used, that of common edges between solutions, but it would be interesting to see how this compares to other standard methods within the literature. All of the suggestions mentioned here would provide more information to a hyper-heuristic and provide the potential for improvement.

- Also in Chapter 5, a hyper-heuristic was proposed which used this measure to select low-level heuristics. The results from using this method showed that the new feature has the potential to add value to hyper-heuristic methods and to improve solution quality. It would seem that this feature warrants further investigation. Currently, it has only been tested in one algorithmic framework. It would be of interest to see the measure be used in an algorithm such as the most successful hyper-heuristic from the *CHeSC* competition, the *AdapHH* of Misir [110], an algorithm which demonstrated the capability to combine multiple measures of heuristic performance.

- It is clear that there is plenty to build on from the new fairness features introduced by Chapter 6. Suggestions are given in the chapter as to how these features can be used to investigate fairness. Specifically the example is given of a 2-depot problem with depots at either end of a 'grid'. In between a fixed number, n, of customers are randomly dispersed. Each depot has a single vehicle. The basic idea is to modify the vehicles' capacities repeatedly starting with 1 vehicle servicing all n vehicles, then changing it so that vehicle serves *n-1* whilst the other vehicle serves 1, and working through all permutations until the first vehicle serves 0 customers and the second vehicle serves all n. For each of these combinations, the solution can be optimised and the objective functions recorded. It could then be observed at which point an increase in fairness causes a large deterioration in solution quality.

- Also concerning the issue of fairness, the feature which modifies the *p-value* in the objective function can be used to encourage a search toward fairness. A study should be performed examining how the value can be adapted through the search to find the correct balance between fairness and solution quality.

- Chapter 6 introduced features which bring the academic vehicle routing problem closer to real-world applications. This work could be continued by introducing further features and constraints into the problem domain. These features may include pickups and deliveries, the need for driver breaks and consideration of traffic and weather conditions.

- It is mentioned above that the VRPTW HyFlex domain introduces 2 new crossover heuristics (see Chapter 3). When the memetic algorithm produces strong results in Chapter 5, it is suggested that the performance of the new crossover heuristics could be a contributing factor. It would be interesting to investigate the strength of these heuristics and whether they have real potential to increase solution quality. To do this, other 'standard' crossover heuristics for the VRPTW should be implemented. The performance of the different methods could then be compared and conclusions drawn.

- Following the algorithmic experiments of Chapter 4, it is discussed that the number of parameters in an algorithm may affect that algorithm's ability to adapt during cross-domain optimisation. This would be an interesting area for further investigation as it has the potential to aid understanding of algorithmic performance when tested on multiple problems. In practical terms, a sophisticated parameter tuning method (such as ParamILS [84]) could be applied to find suitable parameter value for a hyper-heuristic on a set number of problems. A new problem could then be added and the hyper-heuristic tested on this problem. The question would be whether the parameter values still provide strong results or whether they require more tuning.

Bibliography

[1] Enrique Alba and Bernabé Dorronsoro. A hybrid cellular genetic algorithm for the capacitated vehicle routing problem. In *Engineering Evolutionary Intelligent Systems*, pages 379–422. Springer, 2008.

[2] S Alexander. On the history of combinatorial optimization (till 1960). *Handbooks in Operations Research and Management Science: Discrete Optimization*, 12:1, 2005.

[3] Philippe Badeau, François Guertin, Michel Gendreau, Jean-Yves Potvin, and Eric Taillard. A parallel tabu search heuristic for the vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies*, 5(2):109–122, 1997.

[4] Barrie M Baker and MA Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[5] Roberto Battiti, Mauro Brunato, and Franco Mascia. Reactive search and intelligent optimization. 2007.

[6] John E Beasley. Route firstcluster second methods for vehicle routing. *Omega*, 11(4):403–408, 1983.

[7] John E Bell and Patrick R McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18(1):41–48, 2004.

[8] Gleb Belov and Guntram Scheithauer. A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. *European Journal of Operational Research*, 141(2):274–294, 2002.

[9] Russell Bent and Pascal Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893, 2006.

[10] Jean Berger and Mohamed Barkaoui. A new hybrid genetic algorithm for the capacitated vehicle routing problem. *Journal of the Operational Research Society*, 54(12):1254–1262, 2003.

[11] Dimitris J Bertsimas and David Simchi-Levi. A new generation of vehicle routing research: robust algorithms, addressing uncertainty. *Operations Research*, 44(2):286–304, 1996.

[12] Daniel Bienstock, Julien Bramel, and David Simchi-Levi. A probabilistic analysis of tour partitioning heuristics for the capacitated vehicle routing problem with unsplit demands. *Mathematics of Operations Research*, 18(4):786–802, 1993.

[13] Burak Bilgin, Ender Özcan, and Emin Erkan Korkmaz. An experimental study on hyper-heuristics and exam timetabling. In *Practice and Theory of Automated Timetabling VI*, pages 394–412. Springer, 2007.

[14] Kenneth Dean Boese. *Cost versus distance in the traveling salesman problem.* UCLA Computer Science Department, 1995.

[15] Alexandre Le Bouthillier and Teodor Gabriel Crainic. A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers & Operations Research*, 32(7):1685–1708, 2005.

[16] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1):104–118, 2005.

[17] Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. Applying the ant system to the vehicle routing problem. In *Meta-Heuristics*, pages 285–296. Springer, 1999.

[18] Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. An improved ant system algorithm for thevehicle routing problem. *Annals of operations research*, 89:319–328, 1999.

[19] Edmund Burke, Peter Cowling, Patrick De Causmaecker, and Greet Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied intelligence*, 15(3):199–214, 2001.

[20] Edmund Burke, Timothy Curtois, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Sanja Petrovic, JA Vazquez-Rodriguez, and Michel Gendreau. Iterated local search vs. hyper-heuristics: Towards general-purpose

search algorithms. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[21] Edmund Burke, Graham Kendall, D Landa Silva, Ross O'Brien, and Eric Soubeiga. An ant algorithm hyperheuristic for the project presentation scheduling problem. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2263–2270. IEEE, 2005.

[22] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of metaheuristics*, pages 457–474. Springer, 2003.

[23] Edmund K Burke and Yuri Bykov. A late acceptance strategy in hill-climbing for exam timetabling problems. In *PATAT 2008 Conference, Montreal, Canada*, 2008.

[24] Edmund K Burke, Timothy Curtois, Gerhard Post, Rong Qu, and Bart Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2):330–341, 2008.

[25] Edmund K Burke, Timothy Curtois, Rong Qu, and G Vanden Berghe. A time predefined variable depth search for nurse rostering. *INFORMS Journal on Computing*, 2007.

[26] Edmund K Burke, Timothy Curtois, Rong Qu, and G Vanden Berghe. A scatter search methodology for the nurse rostering problem. *Journal of the Operational Research Society*, 61(11):1667–1679, 2010.

[27] Edmund K Burke, Michel Gendreau, Gabriela Ochoa, and James D Walker. Adaptive iterated local search for cross-domain optimisation. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1987–1994. ACM, 2011.

[28] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer, 2010.

[29] Edmund K Burke, Graham Kendall, and Eric Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.

[30] EK Burke, MR Hyde, G Kendall, and JR Woodward. The scalability of evolved on line bin packing heuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation(CEC 2007)*, pages 2530–2537, 2007.

[31] Konstantin Chakhlevitch and Peter Cowling. Hyperheuristics: recent developments. In *Adaptive and multilevel metaheuristics*, pages 3–29. Springer, 2008.

[32] Ching-Yuen Chan, Fan Xue, WH Ip, and CF Cheung. A hyper-heuristic inspired by pearl hunting. In *Learning and Intelligent Optimization*, pages 349–353. Springer, 2012.

[33] Pai-Chun Chen, Graham Kendall, and G Vanden Berghe. An ant based hyper-heuristic for the travelling tournament problem. In *Computational Intelligence in Scheduling, 2007. SCIS'07. IEEE Symposium on*, pages 19–26. IEEE, 2007.

[34] Wen-Chyuan Chiang and Robert A Russell. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on computing*, 9(4):417–430, 1997.

[35] Nicos Christofides and Samuel Eilon. Algorithms for large-scale travelling salesman problems. *Operational Research Quarterly*, pages 511–518, 1972.

[36] G u Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[37] Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997.

[38] Jean-François Cordeau, Michel Gendreau, Gilbert Laporte, Jean-Yves Potvin, and Frédéric Semet. A guide to vehicle routing heuristics. *Journal of the Operational Research society*, pages 512–522, 2002.

[39] Jean-François Cordeau, Gilbert Laporte, Anne Mercier, et al. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, 52(8):928–936, 2001.

[40] Peter Cowling and Konstantin Chakhlevitch. Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, pages 1214–1221. IEEE, 2003.

[41] Peter Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III*, pages 176–190. Springer, 2001.

[42] GA Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[43] T. Curtois, G. Ochoa, M Hyde, and J. A. Vázquez-Rodríguez. A hyflex module for the personnel scheduling problem. Technical report, School of Computer Science, University of Nottingham, 2011.

[44] Tim Curtois. http://www.cs.nott.ac.uk/ tec/nrp/. Website, 2014.

[45] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[46] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[47] Lawrence Davis. Adapting operator probabilities in genetic algorithms. In *International Conference on {G} enetic {A} lgorithms\'89*, pages 61–69, 1989.

[48] Peter Demeester, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice. *Journal of Scheduling*, 15(1):83–103, 2012.

[49] Jörg Denzinger, Marc Fuchs, and Matthias Fuchs. *High performance ATP systems by combining several AI methods.* Citeseer, 1996.

[50] Martin Desrochers, Jan Karel Lenstra, and Martin WP Savelsbergh. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research*, 46(3):322–332, 1990.

[51] Karl F Doerner, Richard F Hartl, and Maria Lucka. A parallel version of the d-ant algorithm for the vehicle routing problem. *Parallel Numerics*, 5:109–118, 2005.

[52] Marco Dorigo, Gianni Caro, and Luca Gambardella. Ant algorithms for discrete optimization. *Artificial life*, 5(2):137–172, 1999.

[53] Marco Dorigo, Vittorio Maniezzo, Alberto Colorni, and Vittorio Maniezzo. Positive feedback as a search strategy. 1991.

[54] John H Drake, Ender Özcan, and Edmund K Burke. An improved choice function heuristic selection for cross domain heuristic search. In *Parallel Problem Solving from Nature-PPSN XII*, pages 307–316. Springer, 2012.

[55] Gunter Dueck. New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational physics*, 104(1):86–92, 1993.

[56] Wout Dullaert and Olli Bräysy. Routing relatively few customers per route. *Top*, 11(2):325–336, 2003.

[57] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2(1):5–30, 1996.

[58] Álvaro Fialho, Luis Da Costa, Marc Schoenauer, and Michèle Sebag. Extreme value based adaptive operator selection. In *Parallel Problem Solving from Nature–PPSN X*, pages 175–184. Springer, 2008.

[59] Marshall L Fisher and Ramchandran Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

[60] Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi. Macs-vrptw: A multiple colony system for vehicle routing problems with time windows. In *New ideas in optimization*. Citeseer, 1999.

[61] Bruno-Laurent Garcia, Jean-Yves Potvin, and Jean-Marc Rousseau. A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Computers & Operations Research*, 21(9):1025–1033, 1994.

[62] TJ Gaskell. Bases for vehicle fleet scheduling. *OR*, pages 281–295, 1967.

[63] Hermann Gehring and Jörg Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99*, volume 2, pages 57–64, 1999.

[64] Hermann Gehring and Jörg Homberger. Parallelization of a two-phase metaheuristic for routing problems with time windows. *Journal of heuristics*, 8(3):251–276, 2002.

[65] Michel Gendreau, Alain Hertz, and Gilbert Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992.

[66] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10):1276–1290, 1994.

[67] Ian P Gent and Toby Walsh. Towards an understanding of hill-climbing procedures for sat. In *AAAI*, volume 93, pages 28–33. Citeseer, 1993.

[68] Billy E Gillett and Leland R Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations research*, 22(2):340–349, 1974.

[69] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[70] Fred Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1):223–253, 1996.

[71] David E Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, 3(5):493–530, 1989.

[72] Bruce L Golden, Arjang A Assad, and Edward A Wasil. Routing vehicles in the real world: applications in the solid waste, beverage, food, dairy, and newspaper industries. *The vehicle routing problem*, 9:245–286, 2002.

[73] Bruce L Golden, Thomas L Magnanti, and Hien Q Nguyen. Implementing vehicle routing algorithms. *Networks*, 7(2):113–148, 1977.

[74] Limin Han and Graham Kendall. Guided operators for a hyper-heuristic genetic algorithm. In *AI 2003: Advances in Artificial Intelligence*, pages 807–820. Springer, 2003.

[75] Limin Han, Graham Kendall, and Peter Cowling. 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 (SEAL02)*, pages 267–271, 2002.

[76] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[77] Wee-Kit Ho, Juay Chin Ang, and Andrew Lim. A hybrid search alogrithm for the vehicle routing problem with time windows. *International Journal on Artificial Intelligence Tools*, 10(03):431–449, 2001.

[78] John H Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[79] Jörg Homberger and Hermann Gehring. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162(1):220–238, 2005.

[80] Jörg Homberger, Hermann Gehring, et al. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *Infor-Information Systems and Operational Research*, 37(3):297–318, 1999.

[81] H Hoos and Thomas Stiitzle. Satllb: An online resource for research on sat. *Sat*, page 283, 2000.

[82] Ping-Che Hsiao, Tsung-Che Chiang, and Li-Chen Fu. A vns-based hyper-heuristic with adaptive computational budget of local search. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

[83] Xiangpei Hu, Qiulei Ding, Yongxian Li, and Dan Song. An improved ant colony system and its application. In *Computational intelligence and security*, pages 36–45. Springer, 2007.

[84] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[85] M Hyde, G Ochoa, T Curtois, and JA Vázquez-Rodríguez. A hyflex module for the boolean satisfiability problem. Technical report, Technical report, School of Computer Science, University of Nottingham, 2009.

[86] Matthew Hyde, Gabriela Ochoa, T Curtois, and JA Vázquez-Rodríguez. A hyflex module for the one dimensional bin-packing problem. *School of Computer Science, University of Nottingham, Tech. Rep*, 2009.

[87] George Ioannou, Manolis Kritikos, G Prastacos, et al. A greedy look-ahead heuristic for the vehicle routing problem with time windows. *Journal of the Operational Research Society*, 52(5):523–537, 2001.

[88] David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[89] Nicolas Jozefowiez, Frédéric Semet, and El-Ghazali Talbi. Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189(2):293–309, 2008.

[90] Soonchul Jung and Byung Ro Moon. A hybrid genetic algorithm for the vehicle routing problem with time windows. In *GECCO*, pages 1309–1316, 2002.

[91] Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28(1):164–191, 1998.

[92] Gerhart F King and Truman M Mast. *Excess travel: causes, extent, and consequences*. Number 1111. 1987.

[93] Joshua D Knowles, Richard A Watson, and David W Corne. Reducing local optima in single-objective problems by multi-objectivization. In *Evolutionary Multi-Criterion Optimization*, pages 269–283. Springer, 2001.

[94] Marek Kubiak. Systematic construction of recombination operators for the vehicle routing problem. *Foundations of Computing and Decision Sciences*, 29(3), 2004.

[95] Marek Kubiak. Distance measures and fitness-distance analysis for the capacitated vehicle routing problem. In *Metaheuristics*, pages 345–364. Springer, 2007.

[96] Gilbert Laporte, Yves Nobert, and Serge Taillefer. Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation science*, 22(3):161–172, 1988.

[97] Jan Karel Lenstra and AHG Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[98] Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal, The*, 44(10):2245–2269, 1965.

[99] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[100] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[101] Helena R Lourenço, Olivier C Martin, and Thomas Stutzle. Iterated local search. *arXiv preprint math/0102188*, 2001.

[102] Silvano Martello and Paolo Toth. *Knapsack problems.* Wiley New York, 1990.

[103] Jorge Maturana and Frédéric Saubion. Towards a generic control strategy for evolutionary algorithms: an adaptive fuzzy-learning approach. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 4546–4553. IEEE, 2007.

[104] Jorge Maturana and Frédéric Saubion. A compass to guide genetic algorithms. In *Parallel Problem Solving from Nature–PPSN X*, pages 256–265. Springer, 2008.

[105] David Meignan, Abderrafiaa Koukam, and Jean-Charles Créput. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879, 2010.

[106] David Mester and Olli Bräysy. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & Operations Research*, 34(10):2964–2975, 2007.

[107] David Mester, Olli Bräysy, and Wout Dullaert. A multi-parametric evolution strategies algorithm for vehicle routing problems. *Expert Systems with Applications*, 32(2):508–517, 2007.

[108] Mustafa Misir, Wim Vancroonenburg, and G Vanden Berghe. A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete. In *Proceedings of the 9th Metaheuristic International Conference (MIC 2011), Udine, Italy*, 2011.

[109] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[110] Mustafa Mısır, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. An intelligent hyper-heuristic framework for chesc 2011. In *Learning and Intelligent Optimization*, pages 461–466. Springer, 2012.

[111] Ahmad Muklason, Andrew J. Parkes, Barry McCollom, and Ender Özcan. Initial results on fairness in examination timetabling. In *The 6th Multidis-*

ciplinary Int. conf. on Scheduling: Theory and Applications (MISTA 2013), 2013.

[112] H Müller-Merbach. Zweimal travelling salesman. *DGOR-Bulletin*, 25:12–13, 1983.

[113] Muhammad Nawaz, E Emory Enscore Jr, and Inyong Ham. A heuristic algorithm for the¡ i¿ m¡/i¿-machine,¡ i¿ n¡/i¿-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[114] Marvin D Nelson, Kendall E Nygard, John H Griffin, and Warren E Shreve. Implementation techniques for the vehicle routing problem. *Computers & Operations Research*, 12(3):273–283, 1985.

[115] Gabriela Ochoa and Matthew Hyde. http://www.asap.cs.nott.ac.uk/external/chesc2011/. Website, 2011.

[116] Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J Parkes, Sanja Petrovic, et al. Hyflex: a benchmark framework for cross-domain heuristic search. In *Evolutionary Computation in Combinatorial Optimization*, pages 136–147. Springer, 2012.

[117] Gabriela Ochoa, James Walker, Matthew Hyde, and Tim Curtois. Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework. In *Parallel Problem Solving from Nature-PPSN XII*, pages 418–427. Springer, 2012.

[118] Institute of Logistics and Distribution Management. The 1985 survey of distribution costs, 1985. Queens Square, Corby, Northants, UK.

[119] Yew-Soon Ong, Meng-Hiot Lim, Ning Zhu, and Kok-Wai Wong. Classification of adaptive memetic algorithms: a comparative study. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(1):141–152, 2006.

[120] I Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking.* PhD thesis, Northweston University, Evanston, IL, 1976.

[121] Ibrahim Hassan Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of operations research*, 41(4):421–451, 1993.

[122] E Özcan, Yuri Bykov, Murat Birben, and Edmund K Burke. Examination timetabling using late acceptance hyper-heuristics. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 997–1004. IEEE, 2009.

[123] Ender Özcan and Ahmed Kheiri. A hyper-heuristic based on random gradient, greedy and dominance. In *Computer and Information Sciences II*, pages 557–563. Springer, 2012.

[124] Ender Özcan, Mustafa Misir, Gabriela Ochoa, and Edmund K Burke. A reinforcement learning-great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing (IJAMC)*, 1(1):39–59, 2010.

[125] Heinrich Paessens. The savings algorithm for the vehicle routing problem. *European Journal of Operational Research*, 34(3):336–344, 1988.

[126] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435, 2007.

[127] Michael Polacek, Richard F Hartl, Karl Doerner, and Marc Reimann. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of heuristics*, 10(6):613–627, 2004.

[128] Marie-Claude Portmann and Antony Vignier. Performances's study on crossover operators keeping good schemata for some scheduling problems. In *GECCO*, pages 331–338, 2000.

[129] Jean-Yves Potvin and Jean-Marc Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, 1993.

[130] Jean-Yves Potvin and Jean-Marc Rousseau. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, 46(12):1433–1446, 1995.

[131] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.

[132] Patrick Prosser and Paul Shaw. Study of greedy search with multiple improvement heuristics for vehicle routing problems, 1996.

[133] Ingo Rechenberg. Evolutionsstrategie94, volume 1 of werkstatt bionik und evolutionstechnik. *Frommann Holzboog, Stuttgart*, 1994.

[134] Marc Reimann, Karl Doerner, and Richard F Hartl. D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563–591, 2004.

[135] Marc Reimann, Michael Stummer, and Karl Doerner. A savings based ant system for the vehicle routing problem. In *GECCO*, pages 1317–1326, 2002.

[136] Gerhard Reinelt. Tspliba traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

[137] Jacques Renaud, Fayez F Boctor, and Gilbert Laporte. An improved petal heuristic for the vehicle routeing problem. *Journal of the Operational Research Society*, pages 329–336, 1996.

[138] Jacques Renaud, Gilbert Laporte, and Fayez F Boctor. A tabu search heuristic for the multi-depot vehicle routing problem. *Computers & Operations Research*, 23(3):229–235, 1996.

[139] Yves Rochat and Éric D Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics*, 1(1):147–167, 1995.

[140] Rubén Ruiz and Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.

[141] Rubén Ruiz and Thomas Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159, 2008.

[142] David M Ryan, Curt Hjorring, and Fred Glover. Extensions of the petal method for vehicle routeing. *Journal of the Operational Research Society*, pages 289–296, 1993.

[143] Martin WP Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA journal on computing*, 4(2):146–154, 1992.

[144] Gerhard Schrimpf, Johannes Schneider, Hermann Stamm-Wilbrandt, and Gunter Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

[145] Jürgen Schulze and Torsten Fahle. A parallel algorithm for the vehicle routing problem with time window constraints. *Annals of Operations Research*, 86:585–607, 1999.

[146] Dale Schuurmans and Finnegan Southey. Local search characteristics of incomplete sat procedures. *Artificial Intelligence*, 132(2):121–150, 2001.

[147] Hans-Paul Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: mit einer vergleichenden Einführung in die Hill-Climbing-und Zufallsstrategie*. Birkhäuser, 1977.

[148] Bart Selman, Henry A Kautz, and Bram Cohen. Noise strategies for improving local search. In *AAAI*, volume 94, pages 337–343, 1994.

[149] Bart Selman, Hector J Levesque, David G Mitchell, et al. A new method for solving hard satisfiability problems. In *AAAI*, volume 92, pages 440–446, 1992.

[150] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint ProgrammingCP98*, pages 417–431. Springer, 1998.

[151] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

[152] Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[153] Éric Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673, 1993.

[154] Éric Taillard, Philippe Badeau, Michel Gendreau, François Guertin, and Jean-Yves Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation science*, 31(2):170–186, 1997.

[155] Eric D Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA journal on Computing*, 6(2):108–117, 1994.

[156] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[157] Kay Chen Tan, Loo Hay Lee, QL Zhu, and Ke Ou. Heuristic methods for vehicle routing problem with time windows. *Artificial intelligence in Engineering*, 15(3):281–295, 2001.

[158] KC Tan, TH Lee, K Ou, and LH Lee. A messy genetic algorithm for the vehicle routing problem with time window constraints. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 679–686. IEEE, 2001.

[159] Xuan Tan, Xuyao Luo, WN Chen, and Jun Zhang. Ant colony system for optimizing vehicle routing problem with time windows. In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, volume 2, pages 209–214. IEEE, 2005.

[160] Sam R Thangiah. *Vehicle routing with time windows using genetic algorithms*. Citeseer.

[161] Sam Rabindranath Thangiah, Kendall E Nygard, and Paul L Juell. Gideon: A genetic algorithm system for vehicle routing with time windows. In *Artificial Intelligence Applications, 1991. Proceedings., Seventh IEEE Conference on*, volume 1, pages 322–328. IEEE, 1991.

[162] Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1539–1546. ACM, 2005.

[163] Dirk Thierens. Adaptive operator selection for iterated local search. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, pages 140–144. Springer, 2009.

[164] Paul M Thompson and Harilaos N Psaraftis. Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations research*, 41(5):935–946, 1993.

[165] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. Siam, 2001.

[166] A Van Breedam. An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints. *Pch. D., University of Antwerp*, 1994.

[167] J. A. Vázquez-Rodríguez, G. Ochoa, T. Curtois, and M Hyde. A hyflex module for the permutation flow shop problem. Technical report, School of Computer Science, University of Nottingham, 2011.

[168] Christos Voudouris and Edward Tsang. Guided local search and its application to the traveling salesman problem. *European journal of operational research*, 113(2):469–499, 1999.

[169] David H Wolpert and William G Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[170] Anthony Wren and Alan Holliday. Computer scheduling of vehicles from one or more depots to a number of delivery points. *Operational Research Quarterly*, pages 333–344, 1972.

[171] PC Yellow. A computational modification to the savings method of vehicle scheduling. *Operational Research Quarterly*, pages 281–283, 1970.

[172] Qiuwen Zhang, Tong Zhen, Yuhua Zhu, Wenshuai Zhang, and Zhi Ma. A hybrid intelligent algorithm for the vehicle routing with time windows. In *Advanced Intelligent Computing Theories and Applications. With Aspects of Theoretical and Methodological Issues*, pages 47–54. Springer, 2008.

[173] Weixiong Zhang, Ananda Rangan, and Moshe Looks. Backbone guided local search for maximum satisfiability. In *IJCAI*, pages 1179–1186. Citeseer, 2003.