Adapting a Hyper-heuristic to Respond to Scalability Issues in Combinatorial Optimisation

by

Richard J. Marshall,
School of Mathematics, Statistics and
Operations Research
Victoria University of Wellington
Supervisors: Dr Mark Johnston
and Prof Mengjie Zhang

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Statistics and Operations Research.

Victoria University of Wellington
2015

Abstract

The development of a heuristic to solve an optimisation problem in a new domain, or a specific variation of an existing problem domain, is often beyond the means of many smaller businesses. This is largely due to the task normally needing to be assigned to a human expert, and such experts tend to be scarce and expensive. One of the aims of hyper-heuristic research is to automate all or part of the heuristic development process and thereby bring the generation of new heuristics within the means of more organisations. A second aim of hyper-heuristic research is to ensure that the process by which a domain specific heuristic is developed is itself independent of the problem domain. This enables a hyper-heuristic to exist and operate above the combinatorial optimisation problem "domain barrier" and generalise across different problem domains.

A common issue with heuristic development is that a heuristic is often designed or evolved using small size problem instances and then assumed to perform well on larger problem instances. The goal of this thesis is to extend current hyper-heuristic research towards answering the question: How can a hyper-heuristic efficiently and effectively adapt the selection, generation and manipulation of domain specific heuristics as you move from small size and/or narrow domain problems to larger size and/or wider domain problems? In other words, how can different hyper-heuristics respond to *scalability* issues?

Each hyper-heuristic has its own strengths and weaknesses. In the context of hyper-heuristic research, this thesis contributes towards understanding scalability issues by firstly developing a compact and effective heuristic that can be applied to other problem instances of differing sizes in

a compatible problem domain. We construct a hyper-heuristic for the Capacitated Vehicle Routing Problem domain to establish whether a heuristic for a specific problem domain can be developed which is compact and easy to interpret. The results show that generation of a simple but effective heuristic is possible.

Secondly we develop two different types of hyper-heuristic and compare their performance across different combinatorial optimisation problem domains. We construct and compare simplified versions of two existing hyper-heuristics (adaptive and grammar-based), and analyse how each handles the trade-off between computation speed and quality of the solution. The performance of the two hyper-heuristics are tested on seven different problem domains compatible with the HyFlex (**Hy**per-heuristic **Flex**ible) framework. The results indicate that the adaptive hyper-heuristic is able to deliver solutions of a pre-defined quality in a shorter computational time than the grammar-based hyper-heuristic.

Thirdly we investigate how the adaptive hyper-heuristic developed in the second stage of this thesis can respond to problem instances of the same size, but containing different features and complexity. We investigate how, with minimal knowledge about the problem domain and features of the instance being worked on, a hyper-heuristic can modify its processes to respond to problem instances containing different features and problem domains of different complexity. In this stage we allow the adaptive hyper-heuristic to select alternative vectors for the selection of problem domain operators, and acceptance criteria used to determine whether solutions should be retained or discarded. We identify a consistent difference between the best performing pairings of selection vector and acceptance criteria, and those pairings which perform poorly.

This thesis shows that hyper-heuristics can respond to scalability issues, although not all do so with equal ease. The flexibility of an adaptive hyper-heuristic enables it to perform faster than the more rigid grammar-based hyper-heuristic, but at the expense of losing a reusable heuristic.

Acknowledgements

Thank you to Dr. Mark Johnston and Prof. Mengjie Zhang, without whose support this thesis would not be possible. The support and help from the members of the Evolutionary Computation Research Group, Victoria University of Wellington is also gratefully acknowledged. I would also like to thank my wife, Irene McKean, for her ongoing love and support which has enabled me to undertake this thesis.

Contents

Chapter 1

Introduction

Determining how to solve an optimisation problem in a new domain, or a specific variation of an existing problem domain, is a task normally assigned to a human expert. In the commercial world, such experts are in short supply and the cost of hiring one is beyond the means of many smaller businesses. Consequently the search for better means of solving new optimisation problem domains rarely moves beyond the boundaries of large corporate organisations and academia. Smaller organisations are often left to satisfy their needs using established rule-of-thumb (heuristic) methods and off-the-shelf solvers such as those supplied within standard spreadsheet software packages (e.g. Microsoft Excel).

When given the task of solving an optimisation problem in a new domain, a human expert will often draw on his or her experience and look for similarities between the new problem domain and an existing problem domain capable of being solved with an established algorithm or heuristic. If a comparable existing problem domain can be identified, then the first attempt at solving the problem in the new domain is often to modify an established algorithm or heuristic to accommodate the different aspects of the new problem domain. Alternatively, the new problem definition is relaxed to match an existing problem domain, which can be solved and the solution repaired to satisfy the aspects of the new problem domain.

This process can be time consuming and result in a complex set of operators that are specific to the problem domain. Any modification to the problem definition may require the whole process to be repeated from the beginning.

Traditional methods of solving combinatorial optimisation problems use algorithms and heuristics, such as a branch-and-bound algorithm [29] or meta-heuristic search, e.g., tabu search [34]. In general, these methods achieve good results but often require detailed domain information and can be complex and time consuming to design and execute. A hyper-heuristic is useful where a more general (domain independent) method is required. A common problem when applying heuristics to an optimisation problem is that they often perform well on some problem instances, but poorly on others. Helping to identify which heuristic to apply to a particular problem instance is one of the objectives of hyper-heuristic research. The hyper-heuristic requires only outline knowledge of the problem domain and is particularly useful when dealing with specific variations to common optimisation problems.

1.1 Hyper-heuristics

The term hyper-heuristic was defined by Cowling et al. [22] as "heuristics to choose heuristics". Ochoa et al. [71] note that the focus in hyper-heuristic research is to adaptively find a solution *method* rather than producing a solution for the particular problem instance at hand. They repeat the observation by Ross [76] that the difference between hyper-heuristics and (meta-)heuristics is that it is the search space of heuristics, rather than the search space of problem solutions, that is traversed. Burke et al. [14] classify hyper-heuristics into two broad categories, those which *select* heuristics and those which *generate* new heuristics by recombining the component operators of one or more existing heuristic(s) (see Figure 1.1). Both categories offer a means to automate the development of heuristics

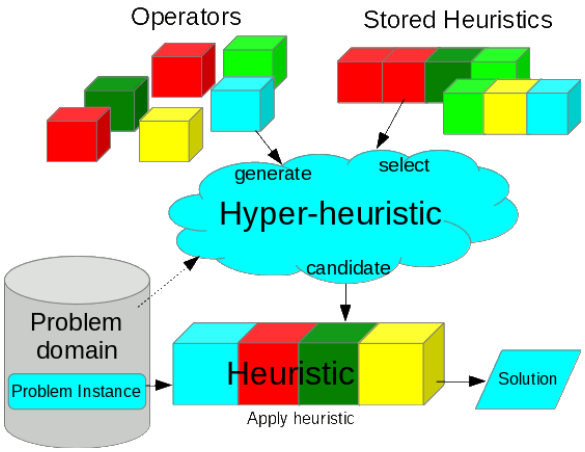

Figure 1.1: The conceptual relationship between a hyper-heuristic, heuristic and operators

to solve problem instances in new problem domains or variations of existing problem domains. Several cross-domain hyper-heuristics have been successfully designed, including those developed by Misir et al. [64] and Sabar et al. [80].

One of the more common hyper-heuristics is Genetic Programming (GP) [44], which is capable of evolving solutions to complex problems and can handle mathematical, logical and operational values with equal ease. By adding a grammar (Grammar Guided Genetic Programming (GGGP) [58]) it is possible to reduce the likelihood of generating semantically meaningless output.

A hyper-heuristic should be suitable for use on any combinatorial optimisation problem domain. To ensure the required level of domain independence is preserved, we focus the main work in this thesis on hyper-heuristics which comply with the HyFlex [70] (**Hy**per-heuristic **Flex**ible) framework specifications. For detailed analysis we use a Vehicle Routing Problem (VRP) domain. The VRP was introduced in 1959 by Dantzig and Ramser [24] and has wide application in transportation and logistics.

1.2 Motivation and Research Questions

The motivation for this research is to reduce the need for scarce, and often expensive, human experts who are currently required when selecting or generating heuristic methods for new, larger or more complex combinatorial optimisation problems. There are potential benefits to industry if an efficient and effective method of solving specific variations to common optimisation problem domains can be automatically evolved in a reasonable development and implementation time and cost.

A common issue with heuristic development is that a heuristic is often designed or evolved using small size problem instances and then assumed to perform well on larger problem instances. The goal of this thesis is to extend current hyper-heuristic research towards answering the question: How can a hyper-heuristic efficiently and effectively adapt the selection, generation and manipulation of domain specific heuristics as you move from small size and/or narrow domain problems to larger size and/or wider domain problems? In other words, how can different hyper-heuristics respond to *scalability* issues?

A problem instance or domain presents scalability issues when the combination of computational time limit, instance size and/or domain complexity requires a hyper-heuristic to alter its default strategy for selecting, generating and manipulating the low-level operators or heuristics in order to deliver a solution within the specified computational time limit. Ultimately the effectiveness of a hyper-heuristic is constrained by the computational time limit and the quality of the unseen low-level operators and heuristics for it to manage. Performance of the hyper-heuristic should therefore be gauged by the degree to which the hyper-heuristic makes effective use of the available computational time rather than the solution the domain specific heuristics and operators deliver.

A hyper-heuristic is provided with only limited knowledge of the problem domain. If this were not the case, then the hyper-heuristic would be

similar to a meta-heuristic and become customised for the specific problem domain. The limited domain knowledge may be restricted to the quantity and broad category of the low-level operators, without any data structure level detail of what action each operator performs or how. Some existing hyper-heuristic frameworks, such as HyFlex (**Hy**per-heuristic **Flex**ible) [70] only allow the hyper-heuristic to extract minimal runtime performance data about each operator and heuristic, requiring the hyper-heuristic to make decisions based on incomplete information.

1.3 Research Approach

We break our investigation into scalability into three stages and develop or adapt suitable hyper-heuristics for each stage. The hyper-heuristics we develop for the second and third stages of this thesis comply with the more restrictive HyFlex framework [70] which provides a clearer definition of the information which may flow across the domain barrier between the hyper-heuristic and the low-level problem domain. HyFlex [70] was developed for the first Cross-domain Heuristic Search Challenge (CHeSC) [69] in 2011. The goals of each stage and the hyper-heuristics we develop are:

1. To develop and use a hyper-heuristic to generate a "compact" heuristic capable of delivering "good" solutions to a range of Capacitated Vehicle Routing Problem (CVRP) instances of different sizes. To achieve this we develop a grammar-based hyper-heuristic using Grammatical Evolution (GE) [78], to generate heuristics for the CVRP domain. This stage is described in Chapter 4.

2. To develop and compare the relative performance, on seven combinatorial optimisation problem domains, of simplified versions of two existing hyper-heuristics. The two hyper-heuristics use very different approaches. One generates new heuristics using a training

set of instances for off-line learning, while the other selects operators (heuristics) using an on-line learning approach. This stage is described in Chapter 5. The two hyper-heuristics used are:

(a) A grammar-based hyper-heuristic using a generic Grammar Guided Genetic Programming (GGGP) method [57]. This is a simplified version of the hyper-heuristic developed by Sabar et al. [80].

(b) A simplified version of the adaptive hyper-heuristic designed by Misir et al. [64] which won the first CHeSC [69] in 2011. This hyper-heuristic uses a single operator selection vector and a new solution is only accepted if its objective value is at least as good as the solution it will replace.

3. To investigate the impact that different features of a problem instance have on the effectiveness of a hyper-heuristic (i.e. *scalability*). We extend the options available to the adaptive hyper-heuristic described above by increasing the number of operator selection vectors and solution acceptance criteria available to it. This final stage is described in Chapter 6.

Although the hyper-heuristics we implement in the second and third stages are capable of generating solutions in any HyFlex [70] compatible problem domain, we use a Vehicle Routing Problem (VRP) domain and instances for detailed testing and analysis before demonstrating the domain independence of the hyper-heuristic on other domains. The VRP has wide ranging application in the transport and logistics industry. Generating a good solution, and possibly several alternatives, to a VRP instance can have significant operational and cost benefits to the industry.

The Capacitated Vehicle Routing Problem (CVRP) [24, 87] contains a single depot holding a fleet of identical vehicles. A set of customers, each at a known location and with a known demand, are to be serviced. The objective is to service all customers while travelling the shortest possible

total distance. Each customer must be serviced only once (split deliveries across multiple routes are not permitted), and the capacity of each vehicle must not be exceeded at any time. This type of problem is relatively easy to understand, but is nevertheless a NP-hard [46] combinatorial optimisation problem.

1.4 Contributions

In the context of hyper-heuristic research, this thesis contributes towards understanding scalability issues in three stages:

1. To develop a compact and effective heuristic that can be applied to other problem instances of differing sizes in a compatible problem domain. This will demonstrate whether a single heuristic can handle scalability issues, or whether different heuristics are required for different sized problems. This stage is detailed in Chapter 4 and summarised in the following articles which have been published, or accepted for publication:

   (a) "Hyper-heuristics, Grammatical Evolution and the Capacitated Vehicle Routing Problem" [55]. This paper was accepted as a poster for the Genetic and Evolutionary Computation Conference (GECCO) in 2014.

   (b) "Developing a Hyper-heuristic using Grammatical Evolution and the Capacitated Vehicle Routing Problem" [54]. This paper was accepted for publication in the proceedings of the 10th Simulated Evolution and Learning (SEAL) conference in 2014.

2. To compare the performance of two different types of hyper-heuristic across seven different combinatorial optimisation problem domains:

   (a) A grammar-based hyper-heuristic which generates a new reusable heuristic from the component parts (operators) of other heuris-

tics. This hyper-heuristic extends the outcomes from the first stage of this thesis.

(b) An adaptive hyper-heuristic which dynamically selects operators from a set of candidates, and develops (on-line) a customised heuristic for an unseen problem instance. The hyper-heuristic is provided with only minimal detail about the operators within the problem domain.

This will establish whether some types of hyper-heuristic respond to scalability issues better or worse than other types of hyper-heuristic. This stage is detailed in Chapter 5 and summarised in the paper "A Comparison between Two Evolutionary Hyper-heuristics for Combinatorial Optimisation" [53]. This paper was accepted for publication in the proceedings of the 10th Simulated Evolution and Learning (SEAL) conference in 2014.

3. To investigate how the adaptive hyper-heuristic developed in the second stage of this thesis can respond, with different computational budgets, to problem instances of the same size, but containing different features and complexity. During this stage we identify which of 48 possible pairings of the key components used by the adaptive hyper-heuristic perform well, and which perform poorly. This stage is detailed in Chapter 6 and summarised in the paper "Hyper-heuristic Operator Selection and Acceptance Criteria". This paper has been submitted to the 15th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP, 2015).

In this thesis we provide a review of relevant literature in Chapter 2 followed by an illustration of how we develop a hyper-heursitic and CVRP problem domain containing a set of relevant operators (heurisitic components) in Chapter 3. Chapters 4, 5 and 6 detail the three stages outlined above, including the experimental results and performance comparisons.

This is followed by overall conclusions and recommendations for further research in Chapter 7.

Chapter 2

Literature Review

This thesis combines three components: the vehicle routing problem, hyper-heuristics, and grammar guided genetic programming. This chapter reviews the literature for each component.

2.1 Combinatorial Optimisation

Many complex everyday problems involve finding an optimal solution in a large, but finite, solution space. Combinatorial optimisation [18] operates on the domain of those optimisation problems, in which the set of feasible solutions is discrete, or can be reduced to discrete, and in which the goal is to find the best solution. Combinatorial optimisation is concerned with the study of effective algorithms and heuristics for solving such problems by intelligently exploring the solution space. In many cases, exhaustive search is not feasible for all but the smallest examples. Many such problems, but not all, are NP-hard [31] problems to solve.

2.1.1 Vehicle Routing Problem

The VRP was introduced in 1959 by Dantzig and Ramser [24], and Lenstra and Rinnooy Kan [46] showed this to be a NP-hard [31] combinatorial op-

timisation problem. The problem domains that are collectively referred to as VRP have been well studied and come in many variations. They have wide ranging application in transportation and logistics.

The aim when solving a VRP is to allocate each customer's delivery to a vehicle route such that the solution achieves a minimum cost of servicing all customers. A VRP includes constraints on the capacity of each vehicle, and/or the duration or distance a vehicle may travel. Further features may be added, such as time windows for servicing a particular customer [82], or allowing multiple depots or interchanges to be considered.

Capacitated Vehicle Routing Problem

One of the simplest variations of VRP is the Capacitated Vehicle Routing Problem (CVRP) [87]. Informally, the CVRP domain can be expressed in the following way. A supplier needs to deliver goods to a number of customers at different locations. Each customer requires a known quantity of goods from the supplier which must be delivered in a single load. To transport the goods, the supplier operates a fleet of identical vehicles. The customer deliveries must be allocated to vehicles such that (a) each customer is visited exactly once, (b) each vehicle's capacity is never exceeded, and (c) the aggregate distance travelled by all vehicles is as short as possible.

In general, a solution for any problem in a VRP domain will consist of a set of vehicle routes, each starting and ending at the depot and visiting a particular sequence of customers. A typical solution to a CVRP instance is given in Table 2.1 and illustrated in Figure 2.1.

Other forms of Vehicle Routing Problem Domains

More complex variations of VRP domains include:

1. *Pick-up and delivery* [1] where goods may need to be transported in either direction between the supplier and the customer or between

Route	Customer sequence (and demand)	Distance (and Load)
1	1, 7 (4), 2 (11), 3 (7), 6 (21), 8 (8), 10 (5), 1	112 (56)
2	1, 11 (6), 9 (1), 4 (8), 5 (14), 12 (12), 14 (13), 1	102 (54)
3	1, 17 (21), 20 (25), 22 (7), 15 (3), 1	77 (56)
4	1, 13 (13), 16 (9), 19 (9), 21 (18), 18 (10), 1	84 (59)
Total		**375 (225)**

Table 2.1: Example CVRP solution. Four routes, starting and ending at the depot (node 1) and visiting selected customers in the sequence shown. Each customer is visited exactly once. Demand (load) shown in brackets. Vehicle capacity is 60. The solution is illustrated in Figure 2.1.

one location and another.

2. *Time windows for delivery* [82, 9] where a customer must be visited between specified times of the day.

3. *Multiple depots and/or satellite facilities* [1] where deliveries may originate from any one of several sources or vehicles replenished mid-route.

4. *Split deliveries* [1] where a delivery for a customer may be divided and delivered by more than one vehicle.

5. *Interchanges* [1] where deliveries in transit may be transferred from one vehicle to another at designated locations. This includes multi-modal transportation problems (e.g. train $\rightarrow$ ship $\rightarrow$ truck).

6. *Non-homogeneous vehicles* [1] where the fleet of vehicles have different capacities and/or capabilities (e.g. refrigerated or bulk liquid transporters).

7. *Open vehicle routing* [1] where a vehicle ends its route at the last customer to be serviced.

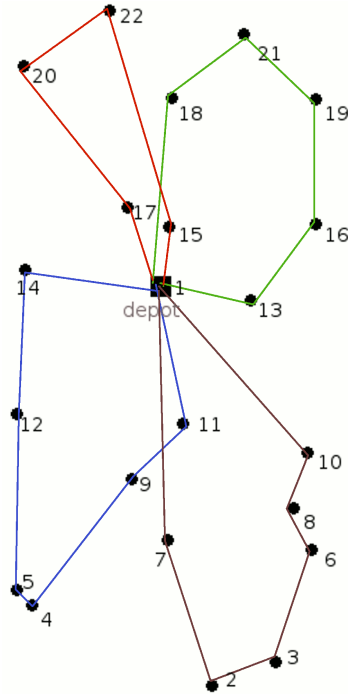

Figure 2.1: Illustration of a 4-route solution to CVRP instance E-n22-k4 [1] shown in Table 2.1 with a total distance of 375.

8. *Capacitated arc routing* [60, 59] where the demand is linked to the arcs between locations rather than the location (customer).

2.1.2 Other Combinatorial Optimisation Domains

The HyFlex [70] framework used in the second and third stages of this thesis contain implementations of other combinatorial optimisation problem domains. These include:

1. **Bin packing**, where the problem is to assign a set of objects to bins, each with a fixed capacity, while minimising the number of bins required [56].

2. **Travelling salesman**, where a single minimum cost route between

locations is required [47].

3. **Permutation flow shop** is a special type of flow shop scheduling problem in which the processing order of the jobs on the machines is the same for each subsequent step of processing (i.e., job B cannot leapfrog ahead of job A later in the sequence if job A was processed ahead of job B early in the sequence). The problem is to sequence the jobs to minimise idle and wait times [84].

4. **Maximum satisfiability (MAX-SAT)**, where the problem is to determine the maximum number of clauses of a given Boolean formula in conjunctive normal form that can be made true by an assignment of truth values to the variables of the formula [3].

5. **Personnel scheduling**, where the problem is to create a roster subject to various constraints [16, 7].

2.1.3 Algorithms and Heuristics for VRP

Cordeau [19], Gendreau et al. [32], Goel and Gruhn [36] and Shaw [81] describe a range of heuristics and meta-heuristics for VRP instances. Their work extends the earlier survey by Laporte [45], who describes six algorithms and four heuristics suitable for solving VRP instances. Solving a VRP instance involves a trade-off between computation speed and achieving the best possible solution. Only relatively small CVRP instances are able to be solved optimally in reasonable computation time (e.g. using branch-and-bound [29]). Consequently heuristics are used to find the best solution possible in the computation time available. In this context, a heuristic is a sequence of operators which develop a solution that is not necessarily an optimal solution, nor guaranteed to be feasible.

There are three established approaches to generating solutions to a CVRP.

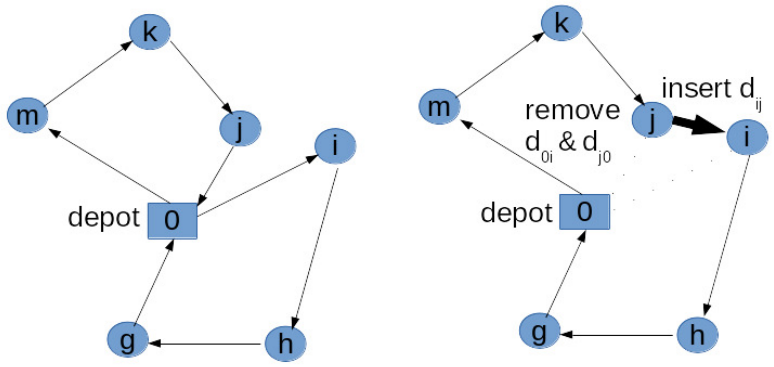

Figure 2.2: Operation of Clarke and Wright Savings heuristic merging two routes into one.

1. *Construction heuristics.* Solutions are built iteratively by inserting and recombining customer deliveries until no further additions are possible. This approach does not attempt to further improve a solution once built. Typical examples include:

   (a) Clarke and Wright Savings (CWS) [17]. This method initially creates out-and-back routes between the depot and each customer. The heuristic combines routes (providing load feasibility is preserved) in a descending order of cost savings, s (see Figure 2.2). The formula used is:

   $$\text{Max } s_{ji} = d_{0i} + d_{j0} - d_{ji}$$

   such that 0 is the depot; i is the first, and j the last, customer serviced on two different existing routes; d_{ij} is the distance between customers i and j.

   (b) Matching based heuristic methods [25, 2] which are similar to CWS but use a different method of calculating the savings value.

   (c) Multi-route improvement heuristic methods [85, 43] which swap or rotate customer deliveries between routes.

A construction heuristic builds a solution step-by-step by selecting and inserting a customer delivery into a route. The selection criteria is typically based on some easy to measure feature of the customer (e.g. nearest/farthest unallocated customer to the depot, or customer with the largest demand). Insertion usually entails examining the existing routes with available capacity and finding the route and place the customer can be serviced with the smallest incremental distance to the current (partial) solution.

2. *Two-phase.* Solutions are developed in two phases in a cluster-first, route-second (or vice versa) process. With this approach, customers are grouped based on some criterion and provisionally assigned to a route constructed from the set of customers in the group. Customers are moved or exchanged between routes until a terminating condition is reached. Typical examples include:

   (a) Fisher and Jaikumar [30] heuristic which uses a solution to a generalised assignment problem to form clusters.

   (b) The Sweep [33] heuristic which clusters deliveries based on the polar co-ordinates of the customer relative to the depot.

   (c) The Petal Method [79] which is an extension of the Sweep method.

3. *Meta-heuristics.* Various methods referred to as meta-heuristics have been developed over the last 40 years. Blum and Roli [8] describe meta-heuristics as strategies that guide the search process with the goal to efficiently explore the adjacent and/or wider solution space in an attempt to find near-optimal solutions. In general, these achieve good results but require a degree of problem specific information and are often complex and time consuming to design and execute. A particularly successful meta-heuristic for solving VRP with time windows [82] is the variation of the Tabu Search heuristic [34] developed by Cordeau et al. [20] who add a simple exchange mechanism

to the search process.

There are a number of variations to Tabu Search [34, 35]. In its basic form, Tabu Search moves from a trial solution to another trial solution by identifying and evaluating the best move from a list of candidate moves. To prevent the search from doubling-back or looping, a move that would restore a previous trial solution is blocked (made "tabu") from being added to the list of candidate moves for a specified number of iterations.

Iterated Local Search (ILS) [48] iteratively alternates between applying an operator (a "kick") which mutates the current solution, and a local search which attempts to improve the mutated solution. The ILS iterations continue until a pre-defined stopping condition is reached, e.g., a time limit. Walker et al. [88] add an adaptive process to a basic ILS which they call Adaptive Iterated Local Search. Their adaptive process uses a learning mechanism to select the mutation and local search operations from a set of candidate operators. The greater the improvement achieved by the operators in a defined number of previous iterations, the more likely those operators will be selected at the start of the next iteration.

Alternative approaches combine the selected features of multiple solutions in an attempt to arrive at a better solution. Such a method is proposed by Nagata and Bräysy [66]. They use a crossover technique they call edge assembly crossover. They report good results using the technique but only demonstrate it on problems with unit demands and parent solutions with the same number of routes.

A general heuristic for a standard CVRP and four variations (with time-windows, multiple depot, non-homogeneous vehicles, and open routing (see Section 2.1.1)), is proposed by Pisinger and Ropke [73]. They firstly reformulate the respective problems into a format they call Rich Pickup and Delivery Problem with Time Windows. Then an adaptive large

neighbourhood search [75] is applied in cycles using neighbourhoods defined by the selected combination of ruin and recreate operators. The ruin and recreate process repeats until no further improvements are found or a pre-defined time limit is reached.

2.2 Evolutionary Computation

Evolutionary Computation (EC) is a sub-field of artificial intelligence that borrows ideas from, and is inspired by, natural evolution and adaptation [90]. EC covers a number of techniques based on evolutionary processes and natural selection, including Ant Colony Optimisation, Evolution Strategy, Genetic Algorithms, Genetic Programming, Learning Classifier Systems, and Particle Swarm Optimisation. This thesis only makes use of Genetic Programming as described below.

2.2.1 Genetic Programming

Genetic Programming (GP) [44, 74] is a population based technique and an extension of both Genetic Algorithms (GA) [21] and automatic programming. The aim of GP is to build computer programs that are not expressly designed and programmed by human beings [52]. Individuals within the population compete for survival by adapting as best they can to the environmental conditions. Each individual is assessed and given a fitness score which determines its relative strength to its peers. This in turn increases or decreases the individual's chance of survival. Copying natural selection, crossovers between individuals, mutation and death are part of the process of adaptation (see Figure 2.3). For each generation, individuals within the previous generation are selected for breeding or, through elitism, transferred unchanged into the new generation. Breeding selection is often done by a tournament selection or roulette wheel process, whereby a predetermined number of individuals are randomly selected,

weighted by fitness. Crossover and/or mutation is performed on the selected individuals as illustrated in Figure 2.3. The process can be a computationally cheap method that is capable of dealing with many problems, providing there is a means of determining the individual's fitness [51]. An advantage GP has over GA is in its ability to handle individuals of different lengths.

2.2.2 Grammar Guided Genetic Programming

Manrique et al. [52] describe Grammar Guided Genetic Programming (GGGP) [57] as an extension of traditional GP by employing context-free grammars (CFG) (see the green shaded box in Figure 2.4) to generate possible solutions to a problem as sentences. GGGP uses the grammar to provide a formal definition of the syntactic problem constraints and uses a derivation tree (see Figure 2.5) for each sentence to encode the solutions. In doing so the likelihood of invalid individuals being generated is reduced.

Grammatical Evolution

Grammatical Evolution (GE) is an evolutionary computation technique pioneered by Ryan et al. [78]. A key feature of GE is the separation between the search engine and the problem. This enables different classes of problem to be solved using the same search engine, and conversely, an alternative engine can be employed to create and evolve a 'genotype'. The linking element is a grammar relevant to the class of problem, e.g., CVRP, which is applied through a mapper to the output of the search engine. Defining a good grammar requires a degree of inspiration and experimentation as the structure and content of the grammar can influence the quality of the result, much in the way the grammar of a natural language defines the richness of that language.

The mapping process is illustrated in Figure 2.4. Each integer in the

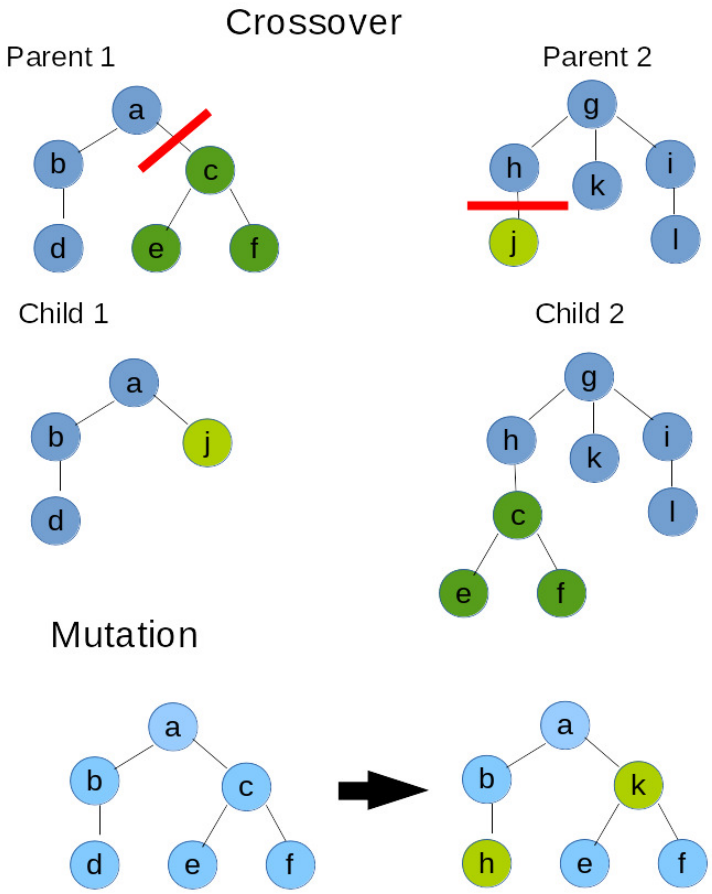

Figure 2.3: Illustration of GP single point crossover and mutation operations on tree-based individuals. The crossover operator takes a branch from each parent and swaps them to produce two new children. The mutation operator randomly transforms individual elements into a new element.

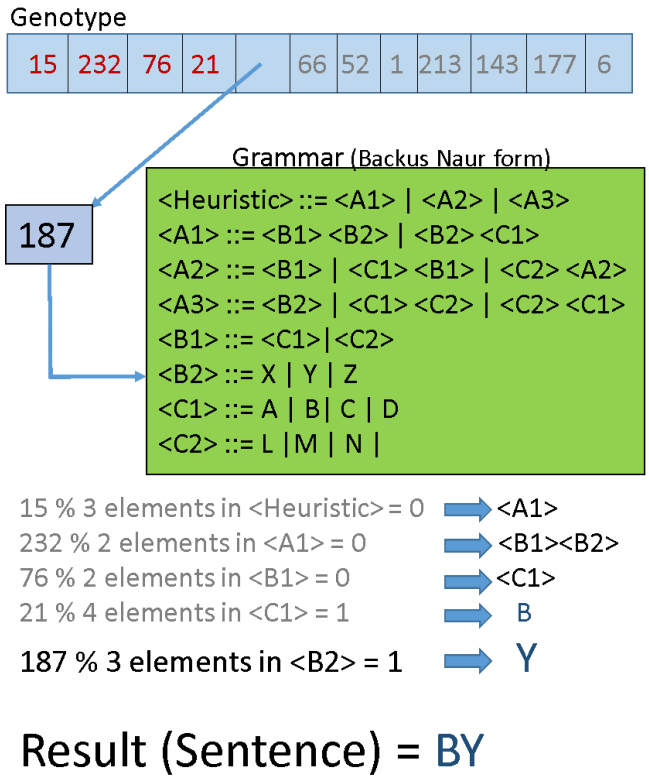

Figure 2.4: A GE linear genotype mapped with a generic Backus Naur form grammar [67] using modular arithmetic. The elements of the genotype are mapped in sequence until a complete sentence is produced.

linear genotype is taken in turn and mapped to the appropriate rule in the grammar. For example, the first integer, 15, is mapped to the first rule in the grammar, <Heuristic>, which has three elements. Using modular arithmetic, 15 mod 3 leaves a remainder of 0, meaning the first element <A1> (a non-terminal) is selected. The second integer in the genotype is mapped to the <A1> rule (2 elements) meaning <B1><B2> is selected. The process continues in a depth-first search until all non-terminals are resolved.

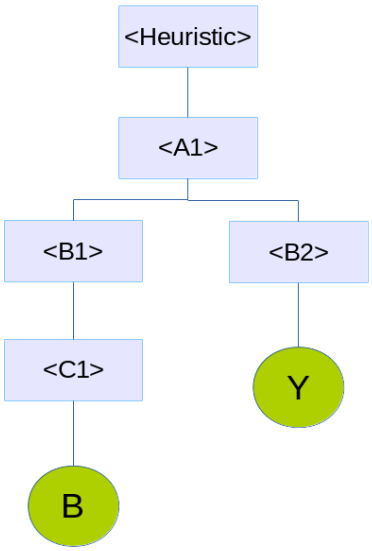

Figure 2.5: A derivation tree created by the example shown in Figure 2.4.

McKay et al. [57] note that GE belongs to a wider family of Grammar Guided Genetic Programming (GGGP) approaches [89] which emerged in the mid 1990s. They note that in GE the genotype is linear, as opposed to the tree structure used in both standard Genetic Programming (GP) [44] and other grammar based GP. The linear genotype enables a range of theory and practice applicable to Genetic Algorithms and Evolution Strategies to be employed. The benefits of GGGP over standard GP include the ability of the grammar to restrict the search space and reduce the likelihood of generating semantically meaningless output.

Over the last decade there has been research into the benefits and challenges of using GE. Rothlauf and Oetzel [77] have investigated the mapping of the genotype to the output (phenotype) in GE and find genotype neighbours in the population do not correspond well with phenotype neighbours produced by the grammar, a phenomenon they refer to as a low degree of locality. The consensus among researchers, notably in the

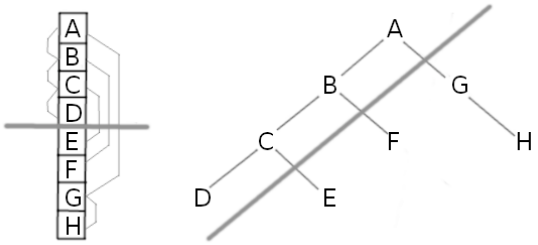

Figure 2.6: The effect of GE Single Point Crossover on a derivation tree. The crossover point between D and E in the linear genotype (left) cuts across three branches of the derivation tree (right).

works of Rothlauf et al. [77, 86], is that there needs to be a close correlation (high locality) between genotype neighbours and phenotype neighbours for an efficient search process. Sub-Tree crossover and Sub-Tree mutation [37] are regarded as the best means of achieving this in GE. This requires a reverse mapping of the phenotype to the genotype to ensure a crossover is only performed between compatible points. Thorhauer and Rothlauf [86] note that the single point crossover operation (see Figure 2.3) has a very different effect when applied in GE than when applied in standard GP, resulting in GE crossovers involving, on average, half of the tree structure as illustrated in Figure 2.6.

Another known feature of GE is redundancy in both the encoding and length of the genotypes. GE uses variable length integer (or bit) strings (codon strings) which are mapped with the grammar using modular arithmetic. This means numerous different encodings will map to the same sentence in the grammar. Also, since the length of genotype can be longer (possibly by a considerable amount) than that needed by the grammar mapper, the population may comprise of many different individuals who only differ from each other in the unused portion of the genotype.

The use of tree-adjunct or tree-adjoining grammars [38] as suggested by McKay et al. [57] and Murphy et al. [65] may remedy some of these

issues, although the technique does not scale well and a grammar such as the one we use in Chapter 4 would generate an unmanageable number of tree elements.

2.3 Hyper-heuristics

As noted in Section 1.1 on page 2, more recent research has looked at hyper-heuristics, which Cowling et al. [22] define as "heuristics to choose heuristics". Ochoa et al. [71] note that the focus in hyper-heuristic research is to adaptively find a solution *method* rather than producing a solution for the particular problem instance at hand. They repeat the observation by Ross [76] that the difference between hyper-heuristics and (meta-)heuristics is that it is the search space of heuristics, rather than the search space of problem solutions, that is traversed. However the delineation between heuristics (rules-of-thumb), meta-heuristics (heuristics employing some form of solutions exploration strategy [8]) and hyper-heuristics can become blurred since both meta-heuristics and hyper-heuristics are also heuristics. In this thesis we treat heuristics and meta-heuristics the same way, and reserve the term hyper-heuristic to mean heuristics to choose heuristics [22].

A hyper-heuristic can be used to dynamically manage the low-level problem domain as the problem instance is being solved (on-line learning), or to develop a heuristic using a separate set of training instances (off-line learning), and apply the resulting heuristic to the problem instance to be solved.

Burke et al. [14] note that one aim of hyper-heuristic research is to provide a general solver for different problem domains. Such a hyper-heuristic is independent of the problem specific domains on which it operates. A hyper-heuristic which achieves this independence is referred to as existing and operating above the domain barrier. Another aim of hyper-heuristic research is to reduce the need for human experts by partially au-

tomating the process which develops and manipulates the low-level (domain specific) operators and heuristics which generate and search through candidate solutions of a problem instance. The desired result from the hyper-heuristic is either a reusable heuristic for the problem domain, or a process which dynamically manipulates low-level operators while solving a given problem instance. The hyper-heuristic needs to deliver a *good* solution in a *reasonable* computation time to a problem instance in any given problem domain. In this respect, the terms *good* and *reasonable* are context sensitive.

Different hyper-heuristic search methods, including the impact of scalability issues, are studied by Keller and Poli [39, 40, 41]. They conclude that using GP to develop a (meta-)heuristic from the component parts of low-level heuristics achieves good outcomes with problem instances of different sizes. Burke et al. [11] study the benefits of adding a memory mechanism to the heuristic design for the one-dimensional bin packing problem to enable basic data (e.g., minimum piece size seen so far) to be stored and recalled as the solution is evolved.

Burke et al. [14] classify current hyper-heuristic approaches into one of two types:

1. **Selection** of one or more heuristics from a small collection of candidate heuristics. The hyper-heuristic first selects the appropriate heuristic(s) and then endeavours to adjust whatever parameters the selected heuristic(s) require to produce a *good* solution. The parameter settings may be dynamically adjusted for the problem instance (on-line learning) or fixed for all instances within a problem domain based on experience from solving a training set of problem instances (off-line learning).

2. **Generation** of new heuristics by recombining the operators (or components) of existing heuristics. An on-line learning process adjusts the operator combinations as well as setting any parameters. An on-

line learning process can also be applied to a set of training instances to determine a reusable sequence of one or more heuristics and any relevant parameters. The evolved sequence of heuristics is stored for future application on unseen problem instances in the same problem domain. If application of the heuristic does not involve further training (e.g. parameter modification), this two-stage combination of training and application is usually referred to as off-line learning. Heuristics evolved in this manner are essentially a recombination of a sequence of operators.

Cross-domain hyper-heuristic approaches have been successfully designed by Misir et al. [63, 64], and Sabar et al. [80] to solve small scale timetabling, bin packing and vehicle routing problem instances of comparable complexity. The hyper-heuristic designed by Misir et al. [64] complies with the HyFlex (**Hy**per-heuristic **Flex**ible) framework [70] specifications. Drake et al. [26] develop a hyper-heuristic which constructs initial solutions to a set of small VRP instances and then modifies the solutions using low-level heuristics within a variable neighbourhood search framework. The mixed quality of the results from this latter work illustrates the challenges faced in choosing the appropriate number and mix of candidate heuristics when using a hyper-heuristic to select or develop new heuristics. Misir et al. [62] also investigate the impact that the choice of candidate heuristics has on the effectiveness of the hyper-heuristic and arrive at a similar conclusion.

Misir et al. [61] summarise some of the various evolutionary and learning methods employed by hyper-heuristic researchers. One of the more common methods is Genetic Programming [44], which is capable of evolving solutions to complex problems and can handle mathematical, logical and operational elements with equal ease. By adding a grammar (Grammar Guided Genetic Programming [58]) it is possible to reduce the likelihood of generating semantically meaningless output. This is particularly useful when evolving sequences of operators containing logical or oper-

ational elements. Burke et al. [14, 10] note that in some respects GP can be regarded as a hyper-heuristic approach to select or generate heuristics. Sabar et al. [80] use GP in this way.

Different applications of hyper-heuristic approaches and problem types are contained in Burke et al. [15] (bin packing using Genetic Programming); Kendall and Li [42] (competitive travelling salesman problem using game theory); and Ochoa et al. [71] (timetabling using a graph based hyper-heuristic approach).

Kendall and Li [42] solve the Competitive Travelling Salesman Problem (CTSP) for two competing salesmen, by providing a selection of five low-level construction operators relevant to the CTSP. These are chosen in a sequence determined by applying Game Theory [68].

Misir et al. [61, 64] develop a hyper-heuristic which generalises across six optimisation problem domains, including vehicle routing. Their work was initially undertaken for the First Cross-domain Heuristic Search Challenge (CHeSC) [69] in 2011, which they won by a comfortable margin against 19 other entrants. The organisers of CHeSC provided the hyper-heuristic framework (HyFlex [70]). The approach by Misir et al. [61, 64] consists of repeating a multi-step process until a pre-determined time limit has expired.

Misir et al. [62] subsequently investigated the impact that different sets of low-level operators have on the performance of the hyper-heuristic. Their hyper-heuristic and subsequent research influences the direction of this thesis. Burke et al. [13] propose a hyper-heuristic approach using Genetic Programming [44] to generate new heuristic methods from operators (or components) of existing methods. Hyper-heuristic research using the Grammatical Evolution [78] variation of GGGP is described in work by Bader-el-Den and Poli [5] and Bader-el-Den, Poli and Fatima [6]. In both cases the problem domain was timetabling and the hyper-heuristic approach used Genetic Programming to generate heuristics.

Two recent works, both using Grammatical Evolution, have a strong

influence on this thesis. Firstly, the hyper-heuristic used by Drake et al. [26] initially constructs a solution to a VRP instance and then seeks to improve the solution using a variable neighbourhood search framework. The grammar used is very detailed and many low level operators and parameters are defined in the grammar. Their work illustrates the many challenges faced when applying a hyper-heuristic. They show that the correct choice of candidate low-level operators and the structure of the grammar are both critical to the outcome.

The second work influencing this thesis is that of Sabar et al. [80]. Their work focuses on using a hyper-heuristic approach to develop a general solver capable of solving a variety of combinatorial optimisation problems including timetabling and VRP. Low-level operators, containing a choice of local search operations, acceptance criteria and adaptive memory parameters are selected and combined to form templates. The hyper-heuristic approach evolves these templates into heuristics relevant to the problem domain. This study also discovered that improved results can be obtained if the evolution process improves multiple solutions in parallel.

In the first and second stages of this thesis we follow the example of Burke et al. [12, 13] and generate new heuristics from the operators (or components) of existing heuristics. Examples of grammars we have used to achieve this are given in Section 4.2 on page 55, and Section 5.2 on page 74. Burke et al. [12, 13] study the evolution of local search methods which define neighbourhoods and efficiently traverse the search space. While application of their evolved methods did not find the best known solutions to the bin packing problems sampled, the technique was shown to work well.

Summary

In this chapter we have reviewed the relevant literature relating to combinatorial optimisation problems (vehicle routing problems in particular),

genetic programming (Grammatical Evolution and grammar-guided GP in particular) and hyper-heuristics. We have discussed the different types (selection and generation) of hyper-heuristic and their application in previous research. In Chapter 3 we discuss the functional requirements of software applications to create hyper-heuristics and to solve generic combinatorial optimisation problem domains. We then develop and implement the applications, which are detailed in Chapters 4, 5 and 6.

Chapter 3

Constructing a Problem Domain and Hyper-heuristic

In this chapter we discuss the features of generic problem domains and hyper-heuristics compatible with the HyFlex [70] framework. We illustrate the features of a problem domain by constructing a Capacitated Vehicle Routing Problem domain. Construction of different types of hyper-heuristic are illustrated in Chapters 4, 5 and 6. The hyper-heuristic and problem domain are linked by the domain barrier which acts as an interface between the two parts (see Figure 3.1).

3.1 Generic Problem Domain

The problem domain contains all the necessary tools to solve a problem instance compatible with that domain, but lacks the knowledge on how to arrive at the best solution. A problem domain needs to provide the following functions to enable compatible problem instances to be solved under the guidance of the hyper-heuristic.

1. The ability to load and interpret a problem instance.
2. To store a set of operators (heuristic components) which can con-

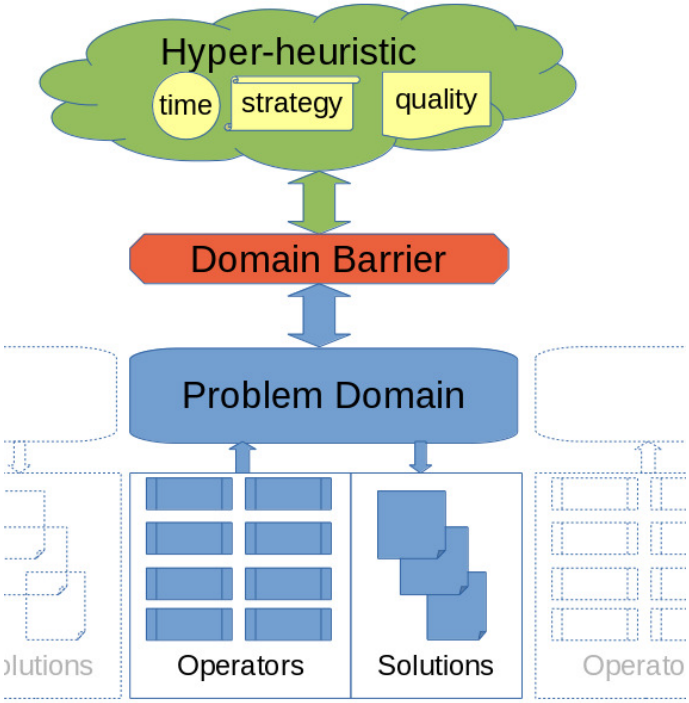

Figure 3.1: High level overview of the linkage between a hyper-heuristic and a problem domain.

struct or manipulate an interim solution to create a new solution. The set should be sufficient in number and diversity to support a variety of actions.

3. The ability to apply an operator to a solution and evaluate the result.

4. To store data relating to the number of times each operator is used while solving a particular problem instance, and the operator's execution time.

5. To generate and store an initial (small) population of solutions.

6. To add or remove a solution from the population of solutions.

7. To calculate a fitness value for a solution and compare it to other solutions. This includes applying any penalty for incomplete or infeasible solutions.

8. To output details of the best solution held in the population of solutions on demand.

In general terms, the hyper-heuristic will instruct the problem domain to apply a particular operator to a selected solution from the population of solutions. Most operators work on a single (primary) parent solution, but some, e.g., a crossover operator, may require a secondary parent solution as well. The problem domain applies the operator and calculates the fitness of the resulting solution. The hyper-heuristic will decide whether to retain or discard the new solution based on acceptance criteria determined by the hyper-heuristic. If the solution is to be retained, then it replaces:

1. The primary parent solution if the new solution is strictly better than the primary parent solution, or if the primary parent solution is **not** the best solution found so far.

2. A randomly chosen solution (other than the best solution found so far) otherwise.

The secondary parent solution (if used) is not altered. A single version of the best found solution so far is always preserved in the population regardless of the acceptance criteria. Should multiple solutions be equally good as the best solution found so far, then only the first best solution found is preserved. All other equally good solutions may be replaced by new, possibly inferior, solutions. Figure 3.2 illustrates this process.

3.2 HyFlex Framework

The HyFlex (**Hy**per-heuristic **Flex**ible) framework [70] was originally developed in 2011 for the First Cross-domain Heuristic Search Challenge

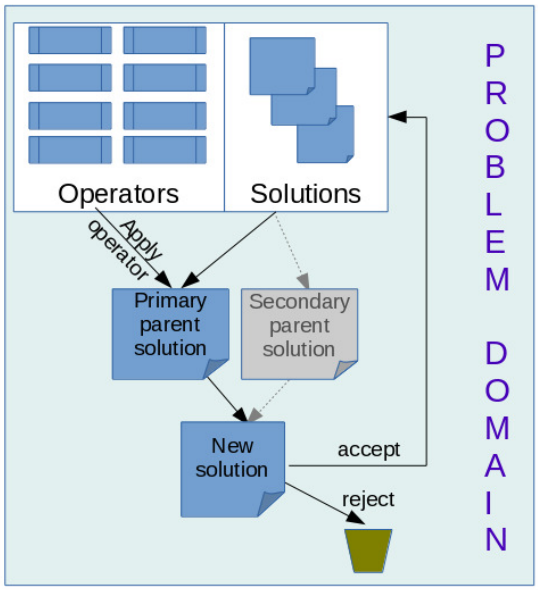

Figure 3.2: An overview of the contents of a generic problem domain and application of an operator to a solution.

(CHeSC) [69]. The framework includes six in-built optimisation problem domains (see Section 2.1.2 on page 14), to which we add a standard Capacitated Vehicle Routing Problem (CVRP) [87] domain:

1. Maximum satisfiability (MAX-SAT)
2. One-dimensional bin packing
3. Permutation flow shop
4. Personnel scheduling
5. Travelling salesman problem (TSP)
6. Capacitated vehicle routing with time windows
7. Standard CVRP (added domain)

Associated with each in-built problem domain is a set of between 8 and 15 unseen low-level operators (heuristics). Each set contains at least one operator belonging to each of the four defined operator types: *mutation*, *ruin-recreate*, *local search* and *crossover*. A crossover operator swaps parts of one solution with another solution in an attempt to create a better solution.

Each operator can use (if appropriate) the two HyFlex parameters α and β, where ($0 \leq \alpha, \beta \leq 1$). The Intensity of Mutation parameter, α, affects the scale of any mutation or ruin operation, e.g., 0.5 would mean half the current solution would be altered by an operator using this parameter. The Depth of Search parameter, β, defines a range or number of repetitions an operator will undertake to find an improved solution in a single execution of the operator.

Each operator is only visible to the hyper-heuristic to the extent allowed by the HyFlex [70] specifications. Operator visibility is restricted to the following properties:

1. **Operator Type.** A mandatory attribute of each operator contained within a HyFlex problem domain. There are four defined operator types:

   (a) **Mutation** operators add or reposition an element in a solution. Operators of this type would generally only involve simple manipulations requiring a short computational time which is only marginally affected by the size of the problem instance.

   (b) **Ruin-Recreate** operators destroy a segment of an existing solution, chosen by the operator implementation, and then rebuild the segment to form a new solution. These operators are more complex than a mutation operator and typically require a longer computational time. The computational time may vary substantially depending on the size of the problem instance.

   (c) **Local Search** operators define and search a solution neighbourhood for improvements. These operators generally apply a de-

gree of logic to the search so can be expected to have a higher chance of improving a solution than the other operator types. However, the computational time may be much longer, and could escalate polynomially (or worse) as the problem instance size increases.

(d) **Crossover** operators combine elements of two current solutions to form a new solution. The computational time of a crossover operator varies but is often similar to a ruin-recreate operator.

2. **Uses Intensity of Mutation.** An indicator to show whether this operator uses the global Intensity of Mutation, α, parameter.

3. **Uses Depth of Search.** An indicator to show whether this operator uses the global Depth of Search, β, parameter.

4. **Call Record.** The number of times the operator has been executed during a run is calculated and is visible to the hyper-heuristic on demand.

5. **Call Time Record.** The aggregate of the execution time of each operator during a run is recorded and is visible to the hyper-heuristic on demand.

3.3 Capacitated Vehicle Routing Problem Domain

We now detail the design of the operators and the management of the population of solutions in our new HyFlex [70] compatible CVRP domain. The domain we have designed includes the ability to easily add or remove individual operators, although we do not use this feature in this thesis.

3.3.1 Operator Design

For the CVRP domain we modify the twelve low-level operators proposed by Walker et al. [88] for a CVRP-with-time-windows domain, by removing the time window elements from each operator. There are 4 mutation, 2 ruin-recreate, 4 local search and 2 crossover operator types (see Section 3.2). Other than each operator's label, type and the use of two global parameters, α and β, the **details** of the operators described in this section are invisible to the hyper-heuristic.

1. **Mutation Operators [M].** All mutation operators move or exchange randomly selected customers. The resulting routes are not necessarily feasible.

   (a) **Swap within route [M0].** A route is randomly selected. Two adjacent customers in that route are randomly selected and their delivery sequence swapped within the route.

   (b) **Move within route [M1].** A route, and two adjacent customers in that route, are randomly selected and the pair randomly moved to elsewhere (not necessarily adjacent) in the delivery sequence within the route.

   (c) **Move to another route [M2].** Two routes are randomly selected. A customer from the first route is randomly selected and inserted into the least-cost position in the delivery sequence of the second route. The operator does not check the modified route for load feasibility.

   (d) **Swap between routes [M3].** Two routes are randomly selected. One customer from each of the two routes is randomly selected and the two customers swapped between the routes. Each customer replaces the other in the sequence of deliveries. The operator does not check the modified routes for load feasibility.

2. **Ruin-recreate Operators [R].** Both ruin-recreate operators destroy a segment of the current solution and reinsert the displaced customers into the solution in the best possible location. Reinsertion positions are restricted to those which result in a feasible route. Both operators use the Intensity of Mutation, α, parameter.

   (a) **Band ruin [R0].** A customer, i, is randomly selected. An additional n customers, where n is the number of customers in the problem instance multiplied by the Intensity of Mutation (α) parameter (rounded up), are also selected by choosing customers with a y-axis coordinate closest to i (see Figure 3.3). The set of $n+1$ customers are removed from their current routes and reinserted in a random order into the route and delivery sequence which provides the least incremental cost of insertion, while retaining load feasibility. A new route is created if necessary.

   (b) **Ring ruin [R1].** A customer, i, is randomly selected. An additional n customers (where n is the number of customers in the problem instance multiplied by the Intensity of Mutation (α) parameter (rounded up)) are also selected by choosing customers with a distance from the depot closest to the distance between the depot and i (see Figure 3.4). The set of $n+1$ customers are removed from their current routes and reinserted in a random order into the route and delivery sequence which provides the least incremental cost of insertion, while retaining load feasibility. A new route is created if necessary.

3. **Local Search Operators [S].** All local search operators use the depth of search, β, parameter. We use the four search operators designed by Walker et al. [88] although none are traditional local search operators.

   (a) **Move if better [S0].** Two routes are randomly selected. A customer from the first route is randomly selected and inserted into

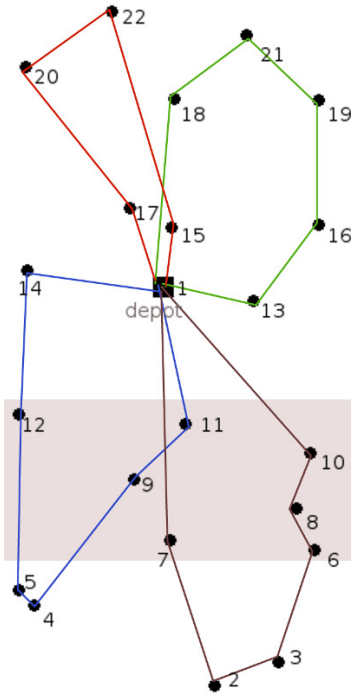

Figure 3.3: Illustration of the zone affected by a band-ruin operator [R0] based on customer number 9 and an Intensity of Mutation, α, value of 0.28

the least-cost position in the delivery sequence of the second route. If the modified second route is infeasible due to overloading, then a single customer (randomly selected from customers with demand $\geq$ excess load) may be displaced from the second route to restore load feasibility. The displaced customer, if any, is reinserted into the least-cost feasible route and delivery sequence, or a new route created if necessary. If the new solution is better than the original solution then the process terminates. Otherwise the process is repeated for up to n iterations, where n is the number of customers in the problem instance multiplied by the Depth of Search, β, parameter (rounded up).

(b) **Swap if better [S1].** Two routes are randomly selected. One

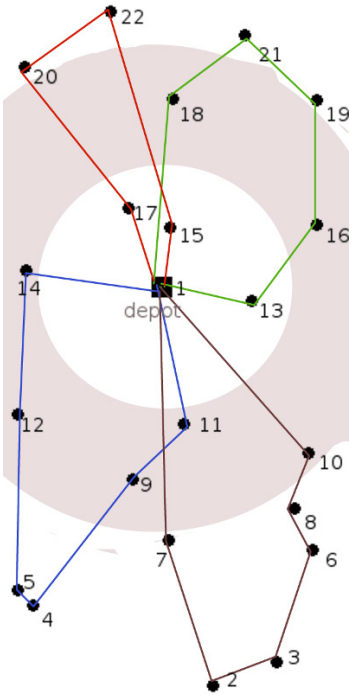

Figure 3.4: Illustration of the zone affected by a ring-ruin operator [R1] based on customer number 9 and an Intensity of Mutation, α, value of 0.28

customer from each of the two routes is randomly selected and the two customers swapped between the routes. Each customer replaces the other in the sequence of deliveries. The resulting routes may not be feasible. If the new solution is feasible and better than the original solution then the process terminates. Otherwise the process is repeated for up to n iterations, where n is the number of customers in the problem instance multiplied by the Depth of Search, β, parameter (rounded up). This operator is similar to the Swap Between Routes operator [M3], but will make up to n attempts to find an improved solution.

(c) **2-opt exchange [S2].** Two routes are randomly selected and

each route is reversed with 0.5 probability. The tails of the two routes are swapped from randomly chosen positions. If the new solution is feasible and better than the original solution then the process terminates. Otherwise the process is repeated for up to n iterations, where n is the number of customers in the problem instance multiplied by the Depth of Search, β, parameter (rounded up).

(d) **Move to best [S3].** Two routes are randomly selected. A customer from the first route is randomly selected and inserted into the least-cost position in the delivery sequence of the second route. If the new solution is feasible and better than the original solution then the process terminates. Otherwise the process is repeated for up to n iterations, where n is the number of customers in the problem instance multiplied by the Depth of Search, β, parameter (rounded up).

4. **Crossover Operators [X].** Crossover operators combine parts of two parent solutions to produce a single new solution.

   (a) **Random combine [X0].** Calculates a value, $v = \frac{(0.5+\alpha)}{2}$ (per Walker et al. [88]), where α is the Intensity of Mutation parameter. Creates a new solution from two solutions by taking individual routes, with probability of v, from the first solution. Then routes from the second solution are added providing there is no duplication of customer deliveries. Any customers not contained in the new solution are inserted in random order into the least-cost route and delivery sequence (or a new route created) in the new solution.

   (b) **Largest combine [X1].** Ranks routes from two solutions based on the number of customer deliveries in the route. Creates a new solution by iteratively adding the ranked routes (largest number of customers first, ties broken randomly) providing no

customer deliveries are duplicated. Any customers not contained in the new solution are inserted in random order into the least-cost route and delivery sequence (or a new route created) in the new solution.

There are no operators belonging to the *other* operator type.

3.3.2 CVRP Solution Generation

The problem domain generates and stores a small population of solutions. Sabar et al. [80] observe that developing multiple solutions in tandem improves the diversity of the population of solutions, and thereby reduces the likelihood of the solution development process stalling. The nature and effect of stalling is discussed further in Section 6.3.1 on page 94. The size of the population is determined by the hyper-heuristic and is a balance between solution diversity and the dilution of computational effort across multiple solutions. If only a single solution is held in the population, then the crossover operators are unable to work correctly and may simply return the parent solution. At the other end of the scale, a large population requires an increase in the time and effort directed towards improving multiple solutions, some of which may be incapable of becoming the best solution.

The CVRP domain we developed generates new solutions by firstly calculating the aggregate demand from all customers, and dividing the aggregate by the vehicle capacity (rounding up) i.e. $\text{routes}_{min} = \left\lceil \frac{\sum \text{demand}}{\text{capacity}} \right\rceil$. This provides the minimum possible number of routes required. The minimum number of empty routes are created and customers are then randomly assigned to a route. Some of the resulting routes may be overloaded and therefore infeasible. Infeasible routes are heavily penalised when the solution fitness evaluation is made to enable all solutions to be evaluated and to encourage retention of feasible solutions.

In the CVRP domain we have developed, feasibility penalties include:

1. **Unallocated customer**: the cost of an out-and-back route from the depot to the customer is added to the fitness value.

2. **Overloaded route**: A manually set penalty plus the cost of out-and-back routes from the depot to each customer allocated to the route is used for the fitness value instead of the total distance of the route.

3. **Duplicated customer**: A manually set penalty. This situation should not occur with the implementation described above.

3.4 Developing a Hyper-heuristic

In this section we describe the requirements to develop a hyper-heuristic compatible with the HyFlex [70] framework. To be effective, a hyper-heuristic needs to be independent of the problem domain. If the hyper-heuristic is not independent of the domain then it should more accurately be described as a meta-heuristic [14].

As discussed in Section 2.1.3 on page 15, Burke et al. [14] classify hyper-heuristics into two broad categories; those which *select* a heuristic from a set of candidate heuristics, and those which *generate* a new heuristic from the operators (components) of existing heuristics. In practise a hyper-heuristic may perform both selection and generation functions at different stages of the process.

3.4.1 Domain Independence

In this thesis we take "domain independence" to include the following features:

1. The hyper-heuristic is linked to the problem domain by an interface (domain barrier, see Figure 3.1) through which only limited standard information may flow. This means the problem domain and hyper-heuristic can be changed without affecting the other.

2. The hyper-heuristic is responsible for controlling the development of a solution to a problem instance by manipulating the operators (heuristic components) within the problem domain. However, the hyper-heuristic has:

   (a) Only outline knowledge of the function each operator performs. It has no knowledge of how the operator performs its function.

   (b) No knowledge about the problem instance size or features.

   (c) No knowledge of the quality of the interim solution other than a single fitness value provided by the evaluation function contained within the problem domain application. The problem domains we use in this thesis seek to minimise the solution fitness.

   (d) Has no means of identifying whether an optimal solution has been achieved.

The HyFlex [70] framework described in Section 3.2 provides a definition of what information may flow across the domain barrier consistent with our description above. There is an assumption that the problem domain application is fit-for-purpose and contains a sufficient number and diversity of operators. Some operators may perform simple add or move functions without applying any logic, while others may be complex heuristics capable of generating a solution to a problem instance on their own. The performance of a hyper-heuristic is constrained by the available computational time, the problem domain complexity and the problem instance size. Since the hyper-heuristic has minimal knowledge about the problem domain and instance, and the computational time limit is usually a manually set parameter, the hyper-heuristic faces a difficult challenge.

Given these constraints, the performance of a hyper-heuristic should be measured on the basis of how efficient it is at using the available computational time. Although the ultimate goal is to enable a problem domain to deliver the best possible solution, it is unreasonable to compare

the quality of a hyper-heuristic to other algorithms and heuristics based on the solution value alone.

3.4.2 Computational Time

Setting the computational time limit is a trade-off between solution quality and speed. When dealing with a new problem domain, or previously unseen problem instance, it is difficult to know an appropriate setting. In part, progress towards achieving a "good" solution is dependent on the quality of the operators built into the problem domain application. These may be limited in their ability to manipulate a solution or require excessive computational time to execute.

A further factor is that the initial solution(s) from which computation begins are controlled by the problem domain and unknown to the hyper-heuristic. This may vary from an empty or randomly generated solution to one which has used an established heuristic to develop a solution of reasonable quality. Further development of the solution(s) from these different starting positions may proceed at widely differing rates.

In the case of an empty or randomly generated solution, a hyper-heuristic can be expected to make significant improvements in the early stages of the computation. As time elapses, however, further improvements become harder to find and an increasingly large number of operator applications fail to make progress. The hyper-heuristic should ideally respond to this situation and we discuss this further in Section 6.3.1 on page 94.

3.4.3 Population of Solutions

Having a population of solutions means the computational time must be divided between different solutions, thereby reducing the time spent working on the "best" solution.

Research by Sabar et al. [80] identified that working on a small number of solutions in tandem achieves a better quality of solution than trying to

improve only a single solution. When using the HyFlex [70] framework, it is preferable for there to be a population of at least two solutions to enable the crossover type of operator to function correctly (see Section 3.2). An upper bound on the number of solutions in the population is a trade-off between maintaining diversity and dividing the available computational time across multiple solutions. In this thesis we use a population of six solutions, which is consistent with the population size used by Sabar et al. [80].

Working with a small number of solutions gives the benefit of focusing computational effort. This increases the number of attempts to improve a particular solution, and can enable a more thorough search for improvements. It also avoids wasting time and effort improving solutions which may be clones of the best found solution, or be so poor that considerable effort is required to improve them.

Working with multiple solutions enables diversity. This can be useful when a particular solution is unable to be improved further using the operators available in the problem domain. Improving an alternative solution from the population may lead to a new best found solution, and unblock the development of better solutions. However, each solution development requires a share of the computational time which can become too thinly spread if the population of solutions is excessive.

Summary

In this chapter we have described the requirements for a generic hyper-heuristic and problem domain compatible with the HyFlex [70] framework, and illustrated construction of a new CVRP domain. In the next chapter we illustrate development of a heuristic using a grammar-based hyper-heuristic.

Chapter 4

Developing a Compact and Effective Heuristic

In this chapter we discuss the first stage of this thesis which has the goal of using a hyper-heuristic to generate a compact and effective heuristic for a problem domain with only deterministic operators. The hyper-heuristic developed for this stage uses the Grammatical Evolution (GE) [78] variation of a grammar guided genetic programming (GGGP) [89] approach. A detailed description of how GE and GGGP work is outside the scope of this thesis. Section 2.2.2 on page 20 gives some background to the relevant aspects of GE and GGGP applicable to this research.

4.1 Hyper-heuristic with Grammatical Evolution

The grammar based hyper-heuristic (GEgrammarHH) used in the first stage of this thesis is developed using Grammatical Evolution (GE) [78]. The concept behind GE, and some of the challenges faced when using it, are discussed in Section 2.2.2 on page 20. In this section we will confine our discussion to the development and application of the grammar and hyper-heuristic used in this chapter.

GEgrammarHH uses a loosely defined interface (domain barrier) be-

tween the hyper-heuristic and the Capacitated Vehicle Routing Problem (CVRP) [87] domain built specifically for this stage. In this respect, it differs significantly from the second grammar-based hyper-heuristic described in Section 5.2 on page 74 and the adaptive hyper-heuristics described in Chapters 5 and 6, which conform to the more rigid HyFlex [70] framework specifications. The hyper-heuristic and CVRP domain described in this chapter were implemented in the GEVA (**G**rammatical **E**volution in Ja**va**) [72] application and our own additional Java program.

One of the aims of this stage is to assess the suitability of GE as a hyper-heuristic, since there are already several reports of using GE for this purposes in the literature (see Section 2.2.2 on page 20).

The CVRP domain used with this hyper-heuristic is described in Sections 4.1.1 and 4.1.2. This domain is different from that described in Section 3.3 on page 36, although it could be adapted to conform to the HyFlex [70] framework specifications. Additionally, this CVRP domain contains only deterministic operators. This means a heuristic applied to a particular CVRP instance will always generate the same solution.

Each heuristic consists of four distinct elements:

1. A $strategy$ which defines how the heuristic is to be developed.

2. A sequence of one or more $operators$ (excluding a search operator) to construct or modify the current partial solution.

3. A $search$ operator to improve the current partial solution.

4. The number of times the whole sequence of operators (including the search operator) is $repeated$ to deliver a complete and feasible solution. We refer to each repetition of the sequence of operators as a $cycle$.

Table 4.1 details the process a generated heuristic will perform with GEgrammarHH.

4.1.1 GEgrammarHH Strategy

The GEgrammarHH enables a choice of strategy when developing a CVRP solution. This is set by the strategy element which defines how the solution is initialised and developed. We include two basic strategies to be employed when developing a heuristic.

1. **Build Strategy:** A heuristic developed using a build strategy starts with an empty solution and routes are iteratively developed. All customers are initially placed in a pool of unallocated customers until selected for placement into the current partial solution. Build operators are used to select and place customers into the solution in a chosen sequence. The placement of customers is such that feasibility of the solution is preserved.

   The strategy may optionally specify that a number of routes be initialised by placing one customer in each route. We refer to this as *seeding*. The method of selection and number of customers allocated during seeding is determined by a parameter of the chosen strategy.

2. **Improve Strategy:** An improvement strategy starts with a complete (i.e. all customers are allocated to a route) and feasible, but possibly sub-optimal, solution developed using a fast heuristic. We use individual out-and-back routes from the depot to each customer, although any fast heuristic (e.g. the CWS heuristic [17]) could be used to develop a starting solution.

4.1.2 GEgrammarHH Operators

An operator manipulates the current partial solution. Application of some operators may result in a customer being returned to the pool of unallocated customers.

A successful heuristic need not necessarily be intuitive, so a range of build, modify and destroy operators are enabled with few constraints on

Table 4.1: Algorithm for GEgrammarHH

$T \leftarrow$ set strategy (*build* or *improve*)
$k \leftarrow$ randomly select number of operators
if T is *build* **then**
  $i \leftarrow$ select initial seeding method
  $s_0^i \leftarrow$ initialise (partial) solution
  for $n \leftarrow 1$ to k **do**
    $op_n^{type} \leftarrow$ select build or improve operator
    $op_n^{repeat} \leftarrow$ select number of repetitions
    $op_n^{param} \leftarrow$ select parameter
    end
  end if
else if T is *improve* **then**
  $s_0 \leftarrow$ initialise (complete) solution (out-and-back routes)
  for $n \leftarrow 1$ to k **do**
    $op_n^{type} \leftarrow$ select improve operator
    $op_n^{repeat} \leftarrow$ select number of repetitions
    $op_n^{param} \leftarrow$ select parameter
    end
  end if
$search^{type} \leftarrow$ select local search operator
$search^{param} \leftarrow$ select parameter(s)
$r \leftarrow 0$
while $(r < 1)$ or $(s_r < s_{r-1})$ or (s_r is incomplete) **then**
  increment r
  $s_r \leftarrow s_{r-1}$
  for $n \leftarrow 1$ to k **do**
    $s_r \leftarrow s_r +$ **execute** op_n^{type}
    end
  $s_r \leftarrow s_r +$ **execute** $search^{type}$
  evaluate s_r
  repeat
return s_{r-1}
end

the selection of an operator or the number of times it is executed. The operators described below, including the search operators, are all deterministic. Consequently, application of a generated heuristic will always produce the same result on a particular problem instance. This is a different approach than that taken with the operators designed for the CVRP domain described in Section 3.3 on page 36, which use random selection of elements.

The CVRP domain contains a set of operators which fall into three categories:

1. **Build operators**. These operators are applied to an empty or partial solution. An unallocated customer is selected and inserted into a route and location providing the route remains feasible and the incremental distance travelled by all vehicles as a result of the insertion is minimised. If insertion cannot be made into an existing route, then a new route is created. The selection of an unallocated customer uses one of the following criteria:

   (a) **Cheapest**: the customer who can be inserted at least incremental cost.

   (b) **Largest Demand**: the customer with the largest demand.

   (c) **Farthest**: the customer farthest from the depot.

   (d) **Nearest**: the customer nearest the depot.

   (e) **Remotest**: the customer farthest away from the depot or any allocated customer.

   In event of ties in the selection process, the customer nearest the depot or with the largest demand (as applicable) is chosen. A build operator has two parameters:

   - How many times execution of the operator is repeated.

- Whether replacement of an existing customer in a route is permitted when considering insertion of a new customer into that route. A replaced customer is blocked from being reinserted in the same route for the remaining repetitions of the current operator. However the customer may be reinserted by another operator or by the same operator in a later cycle.

2. **Improve operators**. These operators perform similar functions to the mutation and ruin-recreate operators described in Section 3.3.1 on page 37. The following operators are enabled:

   (a) **Merge Best Saving**: This operator iteratively takes two routes in the order stored within the problem domain (oldest unmodified route first) and concatenates them by linking the last customer in the first route to the first customer in the second route. During the iterative process the same two routes in opposite order are considered, but the direction of a route is never reversed. If the route is load feasible, and the saving in distance is greater than any other feasible pairing (best-improving), the merger is accepted.

   (b) **Merge Next Nearest**: This operator selects two routes in the same way the *Merge Best Saving* operates. If the route is load feasible, and the distance between the two newly linked customers is shorter than any other feasible pairing, the merger is accepted.

   (c) **Split Routes**: The longest route in the current solution is divided into two. In this thesis the division point is arbitrarily set (to simplify the grammar), but the operator is capable of receiving a parameter to determine the division point. If all routes contain fewer customers than specified by the division point, then this operator has no effect.

(d) **Redo Route**: The route which has remained unaltered for the longest is discarded, and the customers returned to the pool of unallocated customers.

(e) **Remove Lowest Demand**: The allocated customer with the smallest demand is returned to the pool of unallocated customers.

3. **Search operators**. Three search operators are implemented. We limit the search operator to one execution per cycle to keep the computational time within reasonable bounds.

(a) **2 opt**: This search, designed by Croes [23] works on each route in turn and iteratively reverses a segment of the route. The exchange is accepted if a shorter route is achieved as a result, otherwise it is discarded. The search terminates best improving solution.

(b) **3 exchange**: This is similar to 2 opt and iteratively reverses one or both of two adjacent segments of the route. The exchange is accepted if a shorter route is achieved as a result, otherwise it is discarded.

(c) **Iterated Local Search**: This is a deterministic variation of the Iterated Local Search designed by Lourenço et al. [48]. The operation of this search is described in Section 4.1.3. While this search proved successful on the CVRP instances we tested, the data structure and our implementation of the search means it does not scale well and the computational time becomes excessive on CVRP instances larger than 100 customers in size.

4.1.3 Deterministic Local Search

Table 4.2 describes the operation of the deterministic iterated local search algorithm we have developed. The operator requires two parameters to

Table 4.2: Algorithm for Deterministic Iterated Local Search

let N be the set of customers allocated to a route in the current solution
$range \leftarrow$ parameter sets the range of search
pair each i and $j \in N$ **where** $distance_{i,j} \leq range$
sort $pair(i,j)$ by ascending $distance_{i,j}$
while queue of $pair(i,j)$ is not empty
  let R_a and R_b be the routes containing i and j respectively
  if $R_a \neq R_b$ **then** $R_a \leftarrow$ **concatenate** R_a, R_b ($depot$ will appear twice) and R_b removed
  swap positions of i and j in R_a
  improve R_a using either 2opt [23] or 3 exchange
  if $depot$ appears in intermediate position within R_a, divide R_a into two routes
  if better feasible routes result, update current solution.
  repeat
return solution

determine the range of search and whether to use 2 opt [23] or 3 exchange in the second stage. An example is given below.

1. In the example shown in Figure 4.1, we consider the pairing of customers 4 and 6 as part of an iterated local search using a 3 exchange operator. The depot is numbered 0. The two routes are initially (0, 1, 2, **6**, 0) and (0, **4**, 7, 5, 3, 0).

2. These are concatenated to create a route (0, 1, 2, **6**, 0, **4**, 7, 5, 3, 0).

3. The positions of customer 4 and 6 are swapped, producing (0, 1, 2, **4**, 0, **6**, 7, 5, 3, 0).

4. Adjoining segments are individually reversed to search for a better solution. The iteration with the segments (0, 1, 2, | **4**, 0, | **6**, 7, 5, 3, | 0) produces (0, 1, 2, 0, **4**, 3, 5, 7, **6**, 0).

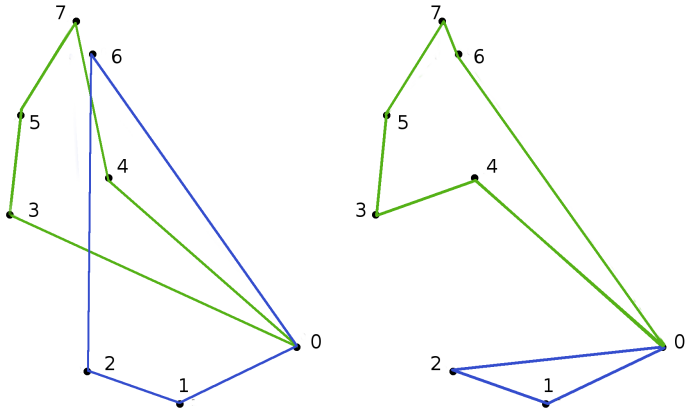

Figure 4.1: Example of the operation of the local search used with GEgrammarHH. Two routes before and after the iterated local search improvements. Other routes and customers omitted for clarity.

5. The route is split into two routes at the position of the interim depot producing [0, 1, 2, 0] and [0, 4, 3, 5, 7, 6, 0]. Evaluation of these two routes gives a better solution than the original solution, so the new solution is retained.

4.2 GE Grammar

We use GE to select the operators and their respective parameters. To achieve this we define a grammar that maps the linear output of the genetic programming search engine (the genotype) to a syntactically correct and semantically meaningful sequence of operators.

The only element of a heuristic that is not specified in the grammar is the number of cycles. Instead, the sequence of operators (up to a predefined maximum number of cycles) are repeated until a cycle ends with:

1. Build Strategy: a solution in which all customers are allocated to a

Table 4.3: Grammar used to develop heuristics for the GEgrammarHH.

<strategy>	::=	build, <seed>, <num>,0 ; <action1> <search>;
<strategy>	::=	improve, 0, 0; <action2> <search>;
<action1>	::=	<build>; | <build>; <action1> | <build>; <action2>
<action2>	::=	<improve>; | <improve>; <action2>
<build>	::=	<select>,<num>,<replace>
<improve>	::=	<improve1>,<num> | <improve2>
<improve1>	::=	mergeBestSavings | mergeNextNearest
<improve2>	::=	splitRoutes, 1, <num> | redoRoute, 1
<improve2>	::=	removeLowestDemand, 1
<search>	::=	2opt | 3exc | iterated-local-search,<num>, 0, <optType>
<select>	::=	cheapest | largestDemand | farthest | nearest | remotest
<seed>	::=	blank | seedcheapest | seedfarthest | seednearest
<num>	::=	1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10
<replace>	::=	0 | 1
<optType>	::=	2 | 3

route.

2. Improve Strategy: no improvements have been made to the solution during the last cycle.

The grammar is structured so a strategy is specified as the first element. Thereafter selected operators, and any required parameters, are added in sequence, terminating with a search operator. The Backus Naur form [67] grammar we use is detailed in Table 4.3.

A typical heuristic from the mapper takes the following form:
build, seedfarthest, 10, 0; mergeBestSavings, 9, 0; cheapest, 5, 1; iterated-local-search, 8, 0, 3;

The heuristic is passed to the problem domain which interprets and processes it (illustrated in Section 4.3). If a complete solution is not achieved

within the pre-defined maximum number of cycles, the distance is set to ∞. The total distance of the solution (fitness) is passed back to the hyper-heuristic.

We adopt the requirement that applying a given heuristic to a particular CVRP instance will always generate the same solution. To this end, all random number generation occurs within the GE search engine and is passed as a parameter(s) with each operator. This includes the local search operator which follows a deterministic sequence when seeking improvements.

4.3 Example of CVRP Solution Construction

Construction of a CVRP solution from an empty solution (*build* strategy) is illustrated in Figures 4.2 through 4.6. This method uses a heuristic generated by a grammar-based hyper-heuristic described in Table 4.3. They are different operators to those described earlier in Section 3.3.1 on page 37.

1. **seedfarthest, 10, 0**. The first operator initialises the solution by creating out-and-back routes to the ten (per the parameter) customers who are the farthest from the depot. An illustration of the result is shown in Figure 4.2. This operator is applied only once during the development of the solution.

2. **mergeBestSavings, 9, 0**. The ten routes created in the previous step are connected together by iteratively concatenating two routes, providing a feasible route is created. A maximum of nine (per the first parameter) concatenations are made. The pairs are linked in a sequence determined by the size of the saving made by the merger. This and the preceding operator are the two operators exhaustively used in the Clarke and Wright Savings heuristic [17] (see Section 2.1.3

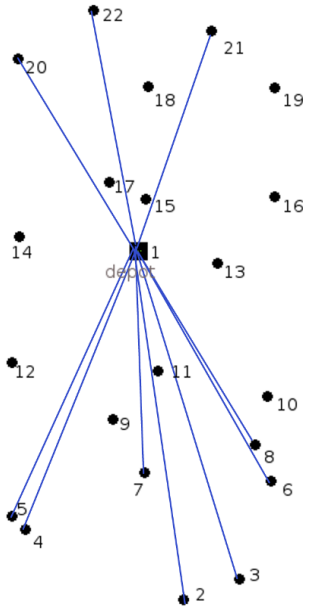

Figure 4.2: Development of a solution to E-n22-k4.vrp [1] problem instance. The solution is initialised by the operator *seedfarthest, 10, 0* by creating out-and-back routes to the ten customers farthest from the depot.

on page 15). An illustration of the result from the application of this operator is shown in Figure 4.3.

3. **cheapest,5,1**. Five (per the first parameter) unallocated customers are added to the interim solution. Customers are selected on a minimum cost of insertion basis (*cheapest*). The second parameter indicates that an existing customer can be displaced from a route if the insertion of the new customer into that route would otherwise make the route infeasible. Displacement is only allowed if the resulting route has a lower cost than before. In the event there is a choice of customers to displace, the one whose removal creates the least cost route is selected. Once displaced, the customer cannot be returned to the same route for the remaining iterations of this operator. In the

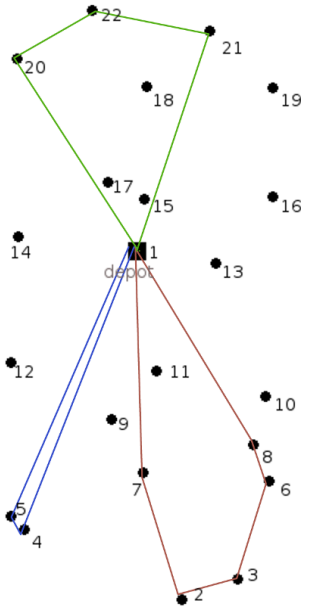

Figure 4.3: In a maximum of nine steps, the ten previous routes are merged to produce three feasible routes.

example illustrated in Figure 4.4 customers 9, 11, 15, 17 and 18 are inserted and customers 21 and 22 displaced.

4. **iterated-local-search, 8, 0, 3**. A local search operator (described in Section 4.1.3) is applied to the interim solution. In this example no improvements are made to the solution shown in Figure 4.4.

Since there are still eight unallocated customers, the solution is incomplete. The heuristic therefore makes another cycle, iteratively applying operators 2, 3 and 4 in each cycle until a cycle ends with a complete solution. In this example the application of operators 2 and 3 in the third cycle add all the remaining customers to the solution (see Figure 4.5). This is the first complete solution and has a total route distance of 493.

The local search operator is applied and improves the solution to that shown in Figure 4.6. The heuristic halts as the solution is now complete

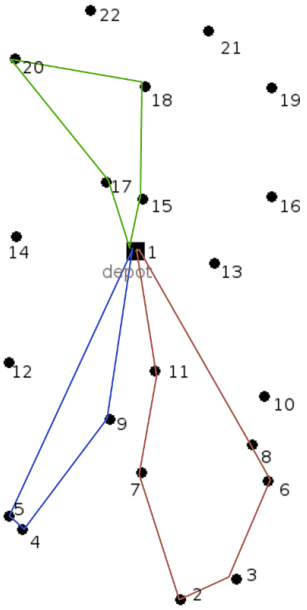

Figure 4.4: The results of the insertion of five new customers (9, 11, 15, 17 and 18) using the *cheapest, 5, 1* operator. The second parameter permits displacement of customers and customers 21 and 22 are returned to the pool of unallocated customers.

(route distance 375). Feasibility has been maintained throughout the development process.

4.4 Experiment Design

We use 30 replications for each of 40 CVRP instances developed by Augerat et al. and Eilon et al. [1, 4, 28]. These instances (from the A and E prefix instances) range in size from 32 to 101 customers (see Tables 4.5 and 4.6) and some contain a variety of features including remote depots, clustering of customers, asymmetric and/or non-Euclidean distances. These instances have been used by other researchers due to their "difficult" nature. We perform six sets of experiments using the combinations of GE

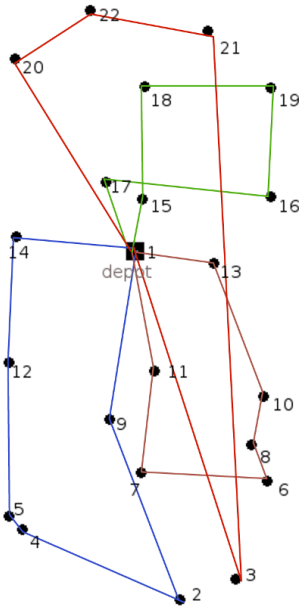

Figure 4.5: All customers are allocated to a route during Cycle 3. This figure shows the interim solution prior to the final application of the local search operator.

population and generations shown in Table 4.4. We test different numbers of GE generations (between 10 and 1,000) and population size (between 40 and 100). Additionally we test different combinations of crossover and mutation functions (see Section 4.5.1).

Each set of experiments is repeated twice, firstly using an on-line learning process to dynamically customise the heuristic for the problem instance being solved, and secondly using off-line learning with a different set (from B prefix instances of Augerat et al. [1, 4]) of six problem instances for training, and applying the resulting heuristic to the 40 CVRP instances. During training the fitness of the heuristic is the aggregate of the distances for each of the six training instances.

In order to evaluate the quality of the generated heuristics, the solutions generated by the best heuristic from each set of experiments are com-

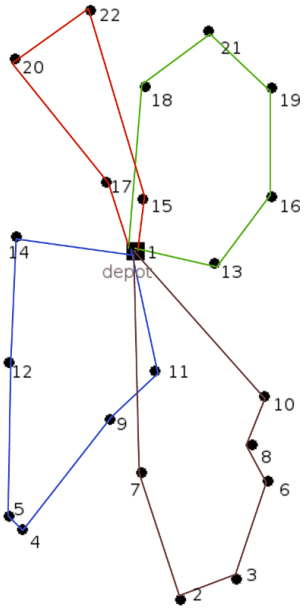

Figure 4.6: The final solution to E-n22-k4.vrp [1] problem instance.

pared to the "best" solution [1] for each problem instance.

4.5 Stage 1 Results

The results from our sixth set of experiments with GEgrammarHH are shown in Tables 4.5 and 4.6 and compared to the "best" results available from [1]. Since some of the published solutions use different distance rounding rules, we have recalculated the published solutions to ensure consistency with the two decimal place rounding of distances used in these experiments. The labelling of the test instances [1] can be identified as being from the set produced by Augerat et al. [4] (prefixed A) or Eilon et al. [28] (E), with n nodes ($n-1$ customers plus a depot) and a minimum of k routes required, i.e. $k = \left\lceil \frac{\sum \text{demand}}{\text{capacity}} \right\rceil$.

Since generation of a CWS heuristic [17] from individual operators is enabled by the grammar, the solution generated by the CWS heuristic

Table 4.4: Grammatical evolution parameter settings. Crossover and mutation functions: single point crossover (SPX), sub-tree crossover (STX), integer flip mutation (IFM) and sub-tree mutation (STM). Average number of operators (NumOps) in a heuristic excludes initialisation and search operators. Quality $= \frac{\sum \text{distance}_{bestHeuristic}}{\sum \text{distance}_{bestPublished}}$

Experiment Set	**1**	**2**	**3**
GE Population	60	100	80
GE Generations	1000	500	100
Crossover rate	0.8	0.9	0.8
Mutation rate	0.2	0.1	0.1
Crossover	SPX+IFM	SPX+IFM	STX+STM
+ Mutation	STX+STM		
Average NumOps	41.3	29.2	16.0
Quality (on-line)	102.3%	102.4%	101.5%
Quality (off-line)	105.2%	104.1%	103.8%
Most common	cheapest	cheapest	farthest
operators	nearest	farthest	cheapest
Least commom	redoRoute	redoRoute	redoRoute
operators	splitRoutes	mergeNextNearest	mergeNextNearest
Experiment Set	**4**	**5**	**6**
GE Population	40	40	40
GE Generations	250	25	10
Crossover rate	0.75	0.8	0.9
Mutation rate	0.25	0.1	0.1
Crossover	SPX+IFM	STX+STM	STX+STM
+ Mutation	STX+STM		
	SPX+STM		
Average NumOps	19.1	11.3	5.4
Quality (on-line)	101.7%	101.0%	100.8%
Quality (off-line)	104.0%	102.4%	101.7%
Most common	cheapest	nearest	cheapest
operators	nearest	largestDemand	farthest
Least commom	redoRoute	redoRoute	redoRoute
operators	splitRoutes	splitRoutes	splitRoutes

Table 4.5: Results (part 1) from on-line learning and off-line learning experiments compared to published best solution [1] (recalculated to ensure rounding consistency). Results from applying the CWS heuristic (see Section 2.1.3 on page 15) shown for comparison.

CVRP	CWS	CWS+2opt	Off-line	On-line	Best
A-n32-k5	843.69	830.67	**787.08**	**787.08**	787.81
A-n33-k5	712.05	712.05	668.66	**662.11**	662.76
A-n33-k6	776.26	776.02	**742.69**	**742.69**	**742.69**
A-n34-k5	810.41	810.41	**780.94**	**780.94**	**780.94**
A-n36-k5	828.47	828.47	810.37	**802.13**	**802.13**
A-n37-k5	707.81	695.42	**672.47**	**672.47**	672.59
A-n37-k6	976.61	976.61	958.66	**950.85**	952.22
A-n38-k5	768.13	766.22	734.18	**733.95**	734.18
A-n39-k5	901.99	901.99	**828.99**	**828.99**	**828.99**
A-n39-k6	863.08	863.07	835.25	**833.20**	**833.20**
A-n44-k7	976.04	972.94	945.44	**938.18**	**938.18**
A-n45-k6	1006.45	1006.45	954.47	949.56	**944.88**
A-n45-k7	1199.98	1199.10	1149.88	**1147.28**	**1147.28**
A-n46-k7	939.74	939.74	918.13	**917.72**	918.13
A-n48-k7	1112.82	1103.99	**1074.34**	**1074.34**	**1074.34**
A-n53-k7	1099.45	1083.81	1020.15	**1012.25**	1013.31
A-n54-k7	1201.20	1187.67	1177.88	**1171.68**	**1171.68**
A-n55-k9	1099.84	1098.41	1100.81	1074.96	**1074.46**
A-n60-k9	1421.88	1410.70	1360.59	**1355.80**	**1355.80**
A-n61-k9	1102.23	1094.49	1051.25	1048.35	**1039.08**

Table 4.6: Results (part 2) from on-line learning and off-line learning experiments compared to published best solution [1] (recalculated to ensure rounding consistency). Results from applying the CWS heuristic (see Section 2.1.3 on page 15) shown for comparison.

CVRP	CWS	CWS+2opt	Off-line	On-line	Best
A-n62-k8	1352.81	1348.53	1303.68	1302 .42	**1294.28**
A-n63-k9	1687.96	1684.02	1638.08	1633.94	**1622.14**
A-n63-k10	1352.48	1343.59	1322.92	1319.05	**1313.74**
A-n64-k9	1486.92	1481.90	1428.14	1415.61	**1400.83**
A-n65-k9	1239.42	1239.03	1192.70	1184.66	**1181.69**
A-n69-k9	1210.78	1205.98	1176.37	**1165.99**	**1165.99**
A-n80-k10	1860.94	1858.98	1792.40	1769.91	**1766.50**
E-n13-k4	**290.00**	**290.00**	**290.00**	**290.00**	**290.00**
E-n22-k4	388.77	388.77	**375.28**	**375.28**	**375.28**
E-n23-k3	621.09	582.93	**568.56**	**568.56**	**568.56**
E-n30-k4	534.45	517.41	**505.01**	**505.01**	**505.01**
E-n31-k7	1263.00	1263.00	1213.00	**1205.00**	**1205.00**
E-n33-k4	843.10	842.80	**837.67**	**837.67**	838.72
E-n51-k5	584.64	584.64	524.81	**524.61**	**524.61**
E-n76-k7	738.13	733.96	693.73	**687.60**	**687.60**
E-n76-k8	794.74	785.63	744.25	**740.66**	**740.66**
E-n76-k10	907.39	902.09	841.42	**836.53**	837.36
E-n76-k15	1054.60	1054.32	1051.63	**1035.81**	**1035.81**
E-n101-k8	889.00	883.97	832.06	**828.84**	**828.84**
E-101-k14	1139.07	1137.91	1108.58	**1095.15**	**1095.15**

should, if discovered by the search engine, represent an upper bound on generated solutions.

Tables 4.5 and 4.6 show that it is possible to apply a GE-based hyper-heuristic to select operators to generate a heuristic for a CVRP instance that delivers a high quality solution. The hyper-heuristic can work with both construction and improvement strategies. The results show that it is possible to develop a heuristic using *off-line* learning that can generally deliver better quality solutions than the CWS heuristic for the range of problem instances used in these experiments. However, the CWS heuristic followed by 2-opt will produce a reasonable result that cannot be consistently matched or beaten by any single heuristic discovered using *on-line* learning. The strength of the on-line learning lies in developing a customised heuristic that produces a better quality solution (compared to off-line learning) to a specific problem instance. Applying the best outcome from a small number of general heuristics developed using off-line learning improves the overall quality of solutions, but still falls short of what can be achieved by generating a customised heuristic for a specific CVRP instance.

Table 4.4 indicates that some of the operators, such as the *splitRoutes* and *redoRoute*, can be simplified or omitted. This table also shows that reducing the number of GE generations leads to the resulting heuristic contains significantly fewer operators. When using 1,000 GP generations, the resulting heuristic can contain a sequence of up to 200 operators, whereas limiting the run to 10 generations means a heuristic rarely contains more than a dozen operators. The heuristics with only a few operators prove to be just as effective as those with numerous operators indicating that increasing the number of operators in a sequence does not necessarily improve performance.

4.5.1 Crossover and Mutation Operators in GE

As part of the experimentation with GEgrammarHH we also examine a number of parameter settings used by GE during the generation of a heuristic. In Section 2.2.2 on page 20 we identify several different ways to perform crossover and mutation in GE. When using the sub-tree crossover and mutation operators recommended by Manrique et al. [21, 52], we observe an early and rapid decline in the diversity of the population (see Figure 4.7). A lack of diversity in the population makes it harder for the search engine to deliver better solutions to those already found.

In our fourth set of experiments (with 250 generations) we compare three different combinations of crossover and mutation operators to observe any change in the performance of the hyper-heuristic and quality of the solutions. As illustrated in Figure 4.7 the performance of these operators is quite different although all eventually arrived at similar solutions despite the diversity (or lack thereof) in the population. The difference between the operators lies in the speed of execution and the average fitness of the current population.

Figure 4.7 illustrates the difference in the performance of the crossover and mutation operators by measuring the average fitness of the individuals in the population in the first 100 generations (of 250) of a typical example (CVRP instance A-n37-k6.vrp [1]). The upper line records the average fitness from using single point crossover and intFlip mutation operators. This results in a wider variance in the fitness of the individuals in the population compared to sub-tree crossover and sub-tree mutation operators (lowest line). The population created by the latter combination rapidly converges towards the best fitness found so far and thereafter shows little diversity. The hybrid single point crossover and sub-tree mutation combination retains diversity in the population for longer, but then converges on the best fitness found so far.

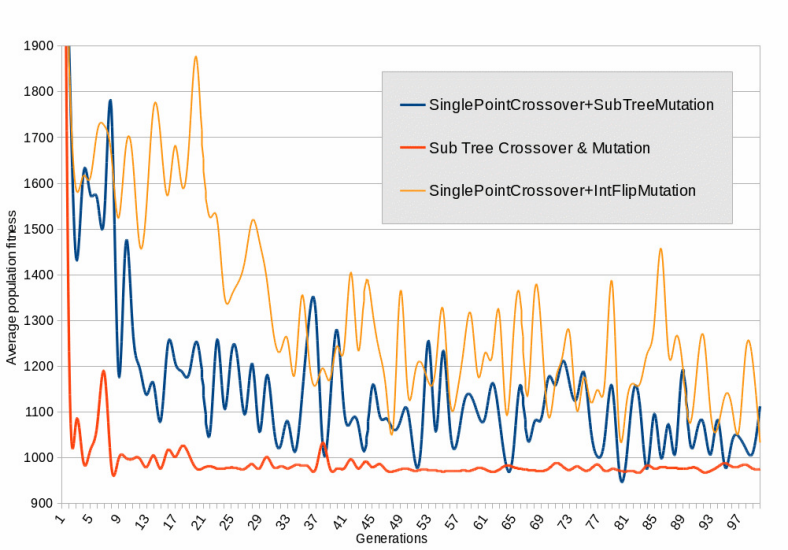

Figure 4.7: Comparison between Sub-Tree and Single Point crossover and Sub-Tree and IntFlip mutation operators. Showing average population fitness per generation on the A-n37-k6.vrp [1] problem instance. Population size = 80.

4.5.2 Identifying Neighbouring Solutions

If, as discussed in Section 2.2.2 on page 20, high locality is a desirable feature, then we need to be able to define what makes two solutions in GE (i.e., heuristics) neighbours. With hyper-heuristics the elements used are operators to be processed rather than the raw data of the problem instance to be solved. If we were solving a CVRP instance directly we could identify neighbouring solutions by the similarity of routes or commonality of arcs between customers.

With a hyper-heuristic, the sequence of operators and parameters cannot be easily identified as neighbours. A minor change to the sequence of operators, or a parameter, will likely result in a radically different solution when the heuristic is applied to a problem instance. It is therefore inappropriate to refer to a sequence of operators as neighbours simply because

they appear similar. This raises the question of whether the feedback from the fitness evaluation of each member of the population needs to be more complex than a single numeric score and, if so, how should the search engine interpret such feedback.

4.6 Stage 1 Conclusions

Although we have not investigated further, improvements to the computational time may be possible when solving larger instances by interpreting the number of repetitions of each operator relative to the problem instance size rather than as an absolute number. This should reduce the number of times the computationally slower search operator is called.

Elements of the hyper-heuristic search process show scope for considerable improvement. The decision to use GE has provided challenges, including many of those discussed in Section 2.2.2 on page 20. While we have achieved good results using GE it is difficult to avoid the impression that the search process is not as efficient as it could be. This inefficiency may lie in the nature of the feedback provided from the fitness evaluation function. A CVRP contains two interdependent sub-problems: vehicle loading and travel distance. The single score based on distance alone does not adequately assist the search engine. Further, as identified by Rothlauf and Oetzel [77], the low degree of locality between genotype and phenotype may hinder an efficient search process. Applying the proposed hyper-heuristic using GE on the CVRP reveals there are a number of barriers to achieving high locality, which we discussed in Section 4.5.2. For this reason, our second grammar based hyper-heuristic (see Section 5.2 on page 74) uses a generic Grammar Guided Genetic Programming (GGGP) approach [89].

Summary

In this chapter we have demonstrated that it is possible to generate both a general and an instance-specific heuristic for CVRP instances using a hyper-heuristic and GE. However, GE is possibly not a good choice of method as a hyper-heuristic and therefore in the following chapters we will apply a more fundamentally sound GGGP method. In the second stage of this thesis we compare the performance of two different hyper-heuristics when applied to multiple problem domains. We modify the hyper-heuristic developed for the experiments described in this chapter to conform to the HyFlex [70] framework and use a more generic grammar.

Chapter 5

Performance Comparison of Different Hyper-heuristics

In this chapter we investigate the second stage of this thesis in which we compare the performance of two different types of hyper-heuristic on seven different problem domains. The two hyper-heuristics are chosen as representative examples of hyper-heuristics at the opposite ends of the selection-generation spectrum [14]. Both hyper-heuristics are compatible with the HyFlex [70] framework (see Section 3.2 on page 33). As discussed in Section 2.3 on page 25, hyper-heuristics can be classified into those which *select* an operator or heuristic from a set of candidate operators, and those which *generate* a new heuristic from the operators (components) of other heuristics. A hyper-heuristic can be used to dynamically manage the low-level problem domain as the problem instance is being solved (on-line learning), or to develop a heuristic using a separate set of training instances (off-line learning), and apply the resulting heuristic to the problem instance to be solved.

In this stage we create two simplified versions of existing hyper-heuristics. These are described in Sections 5.1 and 5.2. The first is an adaptive hyper-heuristic which selects operators dynamically using on-line learning. The second generates a heuristic using off-line learning, and the heuristic is

then applied to the problem instance to be solved. The second hyper-heuristic requires a training phase to generate the heuristic, and we investigate whether the additional time used for training results in a reduced computational time when the heuristic is applied to a problem instance to be solved.

5.1 Adaptive Hyper-heuristic (AdaptiveHH)

An adaptive hyper-heuristic (AdaptiveHH) is similar to a meta-heuristic, and manipulates the unseen low-level operators to find a solution to a single problem instance. AdaptiveHH is only suitable for on-line learning since the process is dynamically customised for a given problem instance.

The original multi-phase adaptive approach by Misir et al. [64], and the modifications made for this thesis, are as follows:

1. **Operator Selection Probability Vector.** Selection of operators is based on a probability vector. In the first phase, all operators have an equal chance of being randomly selected at each iteration. After a defined number of iterations, or time limit, AdaptiveHH starts a new phase and the probability vector is recalculated and normalised based on each operator's success rate, r_i, in the previous phase:

$$r_i = \frac{\text{number of improving solutions}_i}{\text{operator calls}_i}$$

The modified vector improves the chances of operators with a higher success rate being selected during the new phase. Misir et al. [64] use more complex formulae to recalculate the vector, incorporating operator execution time and time remaining. We introduce time weighted performance measures in the third stage of this thesis (see Chapter 6). The selection probability of an operator which consistently fails to improve the solution is gradually reduced to a specified minimum (which may be zero).

2. **Operator Selection and Execution.** To facilitate the crossover operator type (see Section 3.2 on page 33), which uses two parent solutions, a small population of solutions is maintained at any one time. Initially the population consists of, say, 6 solutions generated using the unseen construction method defined for the relevant problem domain. An operator is selected and applied to a solution in the current population of solutions. Misir et al. [64] use the best solution found so far as the primary parent solution whereas we randomly select a parent solution from the population at each iteration.

3. **Reinitialisation.** If no improved solutions are achieved for a specified number of iterations (i_1) then the parameters α, β (see Section 3.2 on page 33) are increased in steps of 0.025. If no improved solutions are achieved for a larger specified number of iterations (i_2, where $i_2 \gg i_1$), then the population of solutions (other than the best found so far) is reinitialised using the construction method defined for the relevant problem domain. The reinitialisation process also resets the parameters α and β to the defined minimum values.

4. **"Relay" Hybridisation.** Misir et al. [64] include a step which selects and executes operators in pairs. We omit this step in the interests of simplicity.

5. **Adaptive Move Acceptance.** In the event no improving solutions are found for a defined number of iterations, Misir et al. [64] incorporate a step which enables selection of a solution other than the best found so far as the primary parent solution for the next iteration. We omit this step as it is unnecessary when randomly selecting the primary parent solution.

The hyper-heuristic is repeated until a pre-determined time limit or early termination condition (i.e. attainment of a target value) is reached.

5.2 HyFlex Compatible Grammar Based Hyper-heuristic

The second hyper-heuristic we describe in this chapter is based on the grammar guided GP [89] approach taken by Sabar et al. [80] which uses GP [44] in the manner described by Burke et al. [10]. The grammar guided genetic programming (GGGP) [89, 57] hyper-heuristic (GrammarHH) is a generation approach using on-line learning during training and no learning when applied to a new problem instance, i.e., it is a generation approach with off-line learning. GrammarHH evolves a single heuristic, with parameter values, suitable for solving comparable problem instances. We use a separate set of six different sized problem instances for training.

GrammarHH develops a reusable heuristic which can be applied to any problem instance in the same domain. The GP population consists of low-level heuristics evolved from component operators in one of the sequences in the "language" defined by the domain independent grammar (see Table 5.1). Each heuristic consists of one or more operators from the *mutation* and/or *ruin-recreate* operator types (see Section 3.2 on page 33) which are applied in sequence. This is followed by a *search* operator which may be preceded or succeeded by a *crossover* operator. This design is similar to the general structure of the heuristics developed in stage one of this thesis.

GrammarHH is implemented in Java using the HyFlex HyperHeuristic [70] template which defines the data structures and methods the hyper-heuristic requires to interface with a HyFlex compatible problem domain. GGGP is implemented as strongly-typed GP within the ECJ [49, 50] Java software package. When evaluating a member of the GGGP population, the evaluation code we have added to the ECJ application creates a GrammarHH object containing the heuristic created from the grammar. GrammarHH implements the heuristic on the unseen problem domain and returns the solution value (fitness) to the ECJ application.

Table 5.1: Grammar used with GrammarHH on the CVRP domain. The HyFlex parameters, α and β, are set by the $< global >$ production rule.

LHS	**Options**
$< heuristic >$	$< global >< alter >< localsearch >$
$< global >$	$< range >< range >$
$< alter >$	$< mutate > \mid < mutate >< alter >$
$< localsearch >$	$< search > \mid < search >< crossover > \mid$ $< crossover >< search >$
$< range >$	$0.2 \mid 0.25 \mid 0.3 \mid 0.35 \mid 0.4 \mid 0.45 \mid 0.5 \mid$ $0.55 \mid 0.6 \mid 0.65 \mid 0.7 \mid 0.75 \mid 0.8$
$< mutate >$	$m0 \mid m1 \mid m2 \mid m3 \mid r0 \mid r1$
$< crossover >$	$x0 \mid x1$
$< search >$	$s0 \mid s1 \mid s2 \mid s3$

Each type of operator in the grammar is identified by a letter prefix followed by a identification number. For convenience, and to avoid use of modular arithmetic, the $< mutate >$, $< crossover >$ and $< search >$ rules in the grammar are customised for each problem domain to match the number of operators of each operator type. A domain independent grammar can be generated by omitting this customisation and using modular arithmetic to map the selected rule to the operator.

Tournament selection is used at each new GP generation to choose the heuristics on which GP crossover and mutation operators are to be applied to create the next generation. Sabar et al. [80] used Grammatical Evolution [78] to manage the GP process, whereas we use a generic Grammar Guided Genetic Programming (GGGP) [89, 57] approach.

During the training phase we apply the evolved heuristic to the same six problem instances used for training during the experiments in stage one (see Section 4.4 on page 60). The fitness value is the aggregate of the

individual solution distances. The evolved heuristic is repeatedly applied to each test problem instance until a time limit or early termination condition is reached.

In these experiments with GrammarHH, only improving or equally good solutions are accepted, whereas Sabar et al. [80] include a choice of eight different acceptance criteria. We repeat training 30 times and store every generated heuristic that delivers a fitness on the six training instances within a specified target threshold (within 1.5% of the aggregate of the best found solutions to each instance).

5.3 Experiment Design

The experiments with GrammarHH were conducted in parallel with AdaptiveHH to measure the comparative performance of the two hyper-heuristics. There is no literature which compares a generation hyper-heuristic and a selection hyper-heuristic on the same problem domain. Our aim is to learn about their relative strengths and weaknesses.

We measure the speed of each hyper-heuristic in finding (if possible) a solution equal to, or better than, one of two targets set at 0.5% and 1.5% above the best solution found during the stage one experiments (see Chapter 4) on each of 58 CVRP instances [1]. These instances are the 40 CVRP instances used in stage one from the A and E prefix instances (see Section 4.4 on page 60) together with 18 B prefix instances of Augerat et al. [1, 4] (in addition to the 6 instances used for training in both stages one and two). The instances range in size between 32 and 262 customers. Due to the small number of instances in the six in-built HyFlex domains, we do not extend the speed-to-target test to those domains.

Both hyper-heuristics will endeavour to select the best performing operators from an unseen set of operators applicable to the problem domain. The problem domains we use contain between 8 and 15 operators. We analyse the frequency with which each operator is selected by Adap-

Table 5.2: Parameters used for experiments.

Parameter	AdaptiveHH	GrammarHH
Time limit	2 mins.	2 mins.
Parameter value range	$0.2 \leq \alpha, \beta \leq 0.8$	$0.2 \leq \alpha, \beta \leq 0.8$
Parameter adjustment threshold	500 calls	n/a
Parameter increment on threshold	0.025	n/a
No progress reinitialisation	10,000 calls	10,000 calls
Hyflex Solution Population	6	6
GP Heuristic Population	n/a	25
GP Generations	n/a	8
GP Crossover probability	n/a	0.8
GP Mutation probability	n/a	0.2

tiveHH in the seven problem domains using fixed two minute runs. In the case of the CVRP domain, we also compare the frequency with the number of times the operator is selected as part of a "good" heuristic generated by GrammarHH.

After some preliminary experimentation we use the parameter settings shown in Table 5.2. Based on the results from the stage one experiments, the GP population size and number of generations are set at the values shown.

5.4 Results and Discussion

We evaluate performance of the operator selection process and the attainment of the solution target values. Our experiments described in Chapter 4 established a best found solution (bfs) for each CVRP instance (minimisation objective) and target values established, being bfs + 0.5% and bfs + 1.5%.

5.4.1 Operator selection and performance

We compare the number of times a particular operator is called, its relative success rate, and failed run rate (i.e. zero improving solutions in a run) when using AdaptiveHH and GrammarHH. We conduct this experiment using a maximum computation time of 2 minutes per run to achieve two target solution values. If, during the run, improving solutions cease to be achieved for 10,000 consecutive operator executions, the population of solutions (other than the best solution found so far) is reinitialised. The run continues with a fresh set of solutions, generated using the unseen construction method defined for the problem domain.

We observe each operator's average execution time but, unlike Misir et al. [64], we do not use the execution time to adjust the operation selection vector. In the CVRP domain the *local search* operators have an execution time approximately 36 times longer, and the *ruin-recreate* and *crossover* operators approximately 12 times longer, than the *mutation* operators on the problem instances used in these experiments.

Table 5.3 records the results from AdaptiveHH across 1,920 runs (30 repetitions $\times$ 64 CVRP instances (58 + 6 GrammarHH training instances)) with the CVRP problem domain using a maximum computation time of 2 minutes per run. We record a *success* against an operator each time the operator delivers an improved solution when executed (which may not be on the best solution to date). A *failed run* is recorded against an operator if the operator fails to deliver a single improved solution during the entire run.

The GrammarHH selection results in Table 5.3 are based on the number of times an operator is contained in the 47 different heuristics that achieved a fitness within a defined threshold (aggregate of best found instance solutions $+1.5\%$) during training.

A total of 724.7 million operator executions were made during the 1,920 runs with AdaptiveHH when allowing a maximum computation time of 2 minutes per run. A run was terminated early if the current solution for

Table 5.3: CVRP Domain: Operator call (selection) and execution record. Operators described in Section 3.3.1. Runs: 1,920 (30 runs $\times$ 64 instances). Total operator calls (AdaptiveHH): 724.7 million. A *failed run* is recorded if an individual operator fails to deliver a single improved solution during an entire run.

Operator (see 3.3.1)	**Calls per Improvement**	**Failed Run**	**AdaptiveHH Call Rate**	**GrammarHH Selection**
M1	10,669	54.9%	3.0%	5.8%
M2	18,117	83.9%	0.9%	1.9%
M3	14,332	42.4%	3.6%	6.7%
M4	18,028	85.6%	0.7%	4.8%
R1	6,727	0.5%	16.1%	15.4%
R2	5,926	0.7%	17.6%	10.6%
S1	4,280	2.5%	16.4%	13.5%
S2	9,688	38.1%	4.4%	8.6%
S3	6,360	20.2%	9.5%	8.6%
S4	6,610	13.6%	8.3%	4.8%
X1	6,572	7.9%	14.0%	13.5%
X2	20,586	0.6%	5.4%	5.8%

the CVRP instance was within a target value of 0.5% above the best solution found during the experiments in the first stage of this thesis. Overall computation time for the 1,920 runs was 30.5 hours.

Comparable operator call results for the six in-built Hyflex problem domains (see Section 3.2 on page 33), grouped by operator type, are shown in Table 5.4. The in-built domains each contain 10 or 12 problem instances. Each problem instance is solved 30 times using different random number generator seeds. The data is therefore based on 300 or 360 runs for each domain.

Table 5.4: Low-level operator type performance across 300 or 360 runs using AdaptiveHH on in-built HyFlex [70] domains. A *failed run* is recorded if an individual operator fails to deliver a single improved solution during an entire run. Note that the in-built crossover operator in the Bin Packing domain fails to capture usage data.

Domain	**Operator Type**	**Number Ops.**	**Call Rate**	**Calls per improvement**	**Failed Run**
CVRP+TW	mutation	3	22.5%	299	29.3%
	ruin-rec.	2	20.2%	503	1.7%
	search	3	39.3%	69	6.2%
	crossover	2	18.0%	229	0.0%
TSP	mutation	5	7.7%	34,482	99.6%
	ruin-rec.	1	1.0%	∞	100%
	search	3	87.1%	5,881	46.0%
	crossover	4	4.2%	75,578	99.9%
MaxSAT	mutation	5	62.4%	88	6.3%
	ruin-rec.	1	3.2%	1,508	81.9%
	search	2	28.3%	84	0.0%
	crossover	2	6.1%	2,565	89.2%
Bin Packing	mutation	3	n/a	146	39.7%
	ruin-rec.	2	n/a	113	11.0%
	search	2	n/a	56	1.5%
	crossover	1	n/a	n/a	35.3%
Flow Shop	mutation	5	8.4%	62,472	99.8%
	ruin-rec.	2	16.9%	770	81.0%
	search	4	68.0%	1,265	5.9%
	crossover	4	6.7%	20,305	99.7%
Psnl Sched	mutation	1	8.9%	∞	100%
	ruin-rec.	3	25.9%	9	76.6%
	search	5	36.8%	2	34.5%
	crossover	3	28.4%	∞	100%

5.4.2 Speed to Target Solution

We assess the speed at which each hyper-heuristic reaches (if possible) two target solution values for each of 58 CVRP instances ranging in size between 32 and 262 customers. A maximum time of 2 minutes is allowed to achieve the target value, after which the run is classified as out of time. Each instance is run 30 times, and the time to reach (if possible) a solution equal to, or better than, the two target values recorded. The two target values are based on the best solution to each CVRP instance found during the stage one experiments. GrammarHH uses the best performing heuristic obtained from the training phase. The results are shown in Table 5.5.

5.4.3 Discussion

Because AdaptiveHH adjusts the probability of an operator being selected based on its performance, selections become biased towards operators which have a higher success rate. Table 5.3 illustrates that both hyper-heuristics select operators in roughly the same proportion. However, AdaptiveHH uses the full range of operators during each run, whereas GrammarHH uses only one generated heuristic (of 47) containing a small subset of operators. The low success rate of mutation operators is offset by the relative speed of execution. However, some mutation operators have a very high failed run rate meaning additional calls of this type of operator may not achieve improving solutions regardless of the number of calls made. By comparison, the two ruin-recreate type operators in the CVRP domain have a relatively high success rate and a very low failed run rate, suggesting additional calls of either of these two operators might achieve improved solutions faster. The call rate shown in Table 5.3 illustrates that AdaptiveHH has progressively biased the selection towards the ruin-recreate operators and the first of the local search operators.

Table 5.4 illustrates that operators of a particular type do not perform consistently across domains and in some cases individual operators may

Table 5.5: Time to achieve (if possible) target solution on 58 CVRP instances run 30 times. Maximum 2 minute time limit. Target based on best found solution (bfs) to each instance during preliminary test runs.

Time (seconds)	**Adaptive bfs** $+ 0.5\%$	**Grammar bfs** $+ 0.5\%$	**Adaptive bfs** $+ 1.5\%$	**Grammar bfs** $+ 1.5\%$
$0 \rightarrow 1$	491	266	726	453
$1 \rightarrow 5$	150	192	228	311
$5 \rightarrow 10$	93	98	104	138
$10 \rightarrow 20$	91	94	101	122
$20 \rightarrow 30$	43	47	60	50
$30 \rightarrow 60$	71	77	92	91
$60 \rightarrow 120$	99	87	88	102
out of time	702	879	341	473
≤ 5 secs.(all)	37%	26%	55%	44%
.. small inst.	79%	62%	94%	79%
.. mid size inst.	48%	35%	68%	62%
.. large inst.	6%	4%	26%	20%
out of time (all)	40%	51%	20%	27%
.. small inst.	11%	23%	1%	8%
.. mid size inst.	34%	47%	15%	20%
.. large inst.	71%	77%	40%	50%

have a very high failed run rate (100% in some cases). This leaves some domains with only a few productive operators, e.g., the Flow Shop domain. The challenge for the hyper-heuristic is to quickly identify and focus on applying those operators which perform well. The call rate results in Table 5.4 indicate that AdaptiveHH has, in general, successfully identified the best performing operators and biased selection accordingly.

Generally, AdaptiveHH outperforms GrammarHH in both speed and number of times the target value is attained. This relative performance

between the two hyper-heuristics is consistent across all sizes of problem instances. Repeat experiments using the next three best heuristics developed during GrammarHH training achieve similar outcomes. Allowing more computational time enables some out-of-time runs to achieve the target value.

Overall, knowledge gained during the training phase of GrammarHH (i.e. which domain-specific operators are productive) does not improve the computation speed compared to AdaptiveHH when working on a problem instance to be solved. The computational time expended during the training phase is therefore not recovered. There is evidence that the flexibility of AdaptiveHH outperforms the more rigid structure of GrammarHH when applied to new problem instances. This is possibly due to the variable performance of the mutation operators, which periodically fail for the entire run. In such cases, AdaptiveHH applies alternative operators, whereas GrammarHH is unable to adjust its approach. However the difference in performance may also be due to numerous factors ranging from the structure and content of the GrammarHH grammar to the arbitrary setting of the parameters both hyper-heuristics require. The fitness function used during training may also have biased the heuristic's fitness evaluation towards larger problem instances.

Deciding whether to terminate a run early, or allow execution to continue until the time limit is reached, is a trade-off between speed and quality. In this stage we use attainment of a target value to trigger early termination. However it may not always be possible to specify a realistic target when dealing with previously unseen problem instances. As shown in Table 5.5, some runs find a "good" solution in under 1 second. With AdaptiveHH at least one (in some cases, all) of the thirty runs with each instance attained the target solution within one second with 50 of the 58 CVRP instances.

5.5 Stage 2 Conclusions

These experiments indicate that both hyper-heuristics can successfully manipulate unseen low-level operators in different problem domains to deliver solutions of a reasonable quality. Table 5.4 shows that operators can vary substantially in effectiveness and efficiency, requiring the operator selection process to respond to domain specific factors. The stage two experiments show the flexibility of AdaptiveHH outperforms the more rigid structure of GrammarHH in both computational speed and solution quality.

With hindsight, there appears to be a good case for including the operator execution time in the operator selection vector adjustment process when using AdaptiveHH. Also, increasing the number and range of choice of Operator Selection Vectors and Solution Acceptance Criteria may also provide opportunities for better performance.

Summary

In this chapter we have compared the performance of two hyper-heuristics of contrasting design. As discussed, extending AdaptiveHH to provide a wider choice of Operator Selection Vectors and Solution Acceptance Criteria may improve the ability of the hyper-heuristic to respond to scalability issues. We investigate this in the third stage of this thesis, which is described in Chapter 6.

Chapter 6

Managing Scalability with an Adaptive Hyper-heuristic

As discussed in Chapter 5, we show that an adaptive hyper-heuristic can be a successful approach when applying an evolutionary computation method to solving combinatorial optimisation problems. By using a pairing of an operator (heuristic) selection vector and solution acceptance criteria, an adaptive hyper-heuristic can manage development of a "good" solution within an unseen low-level problem domain in a commercially realistic computational time. However not all selection vectors and solution acceptance criteria pairings deliver competitive results when faced with differing problem instance features and computational time limits. In this stage of this thesis we use pairings of six different operator selection vectors and eight solution acceptance criteria, and monitor the performance of the adaptive hyper-heuristic when applying each pairing to a set of 50 CVRP instances we have created. The problem instances are all the same 80-node size (79 customers + 1 depot), but with different features. Our results show that a few pairings of operator selection vector and acceptance criterion perform consistently well, while others require a longer computational time to deliver competitive results. We also investigate some of the features of a problem instance that may influence the performance of

the selection vector and acceptance criterion pairings.

The adaptive hyper-heuristic, AdaptiveHH2, described in this chapter is an enhanced version of the adaptive hyper-heuristic described in Chapter 5. With AdaptiveHH2 we provide multiple choices for Operator Selection Vector and Solution Acceptance Criteria. Conceptually, AdaptiveHH2 iteratively selects and applies an unseen operator from the problem domain. The resulting solution is then retained or discarded based on the acceptance criteria specified by AdaptiveHH2. AdaptiveHH2 requires a number of parameters which set the computation time, the number of intermediate decision points (phases), and the choice of operator selection vector and acceptance criterion to use (see Figure 6.1). The two main components of AdaptiveHH2 are:

1. **The Operator Selection Vector**. This vector is used to select the next operator to apply. The vector is updated at the start of each phase based on the performance of the operator in the preceding phase(s). It determines the probability of each operator being selected and applied to the current solution.

2. **The Solution Acceptance Criteria**. Once an operator modifies a solution to create a new solution, the hyper-heuristic needs to decide whether to accept (retain) or discard the new solution.

6.1 Operator Selection Vector Design

AdaptiveHH2 operates for a specified time limit which is broken down into a specified number of phases of equal duration. The operator selection vector is updated at the end of each phase. The choice of selection vector and acceptance criterion is fixed at the beginning of the run and is not altered during the run. The selection vector consists of an array of operators, each with a probability of selection. In the initial selection vector (regardless of type) all operators have an equal probability of selection.

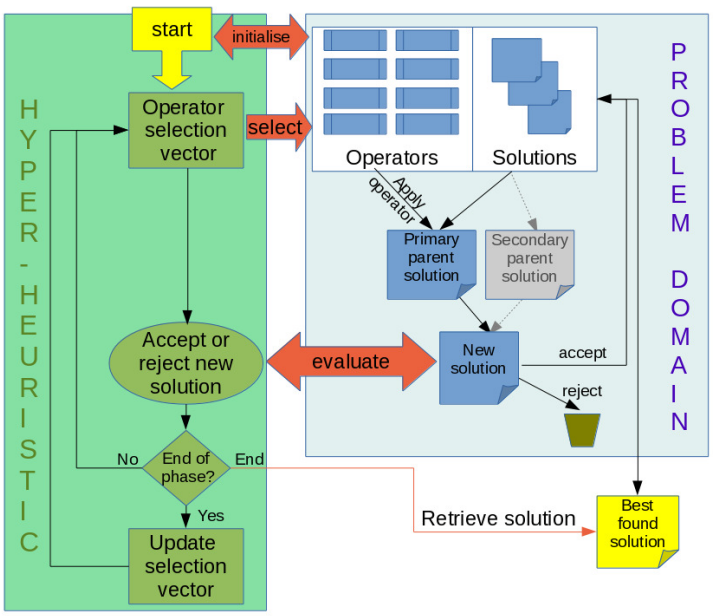

Figure 6.1: Overview of how an adaptive hyper-heuristic interacts with a low-level problem domain across the domain barrier.

We follow the example of Misir et al. [64] and allow some of the selection vectors described below to exclude operators for one or more phases (i.e. the selection probability is zero). The number of phases an unsuccessful operator is excluded is based on a performance penalty. The first time an operator is excluded, the performance penalty is set to one. This means the operator is readmitted to the selection vector at the end of the next phase (i.e. one phase exclusion) with a probability of 0.01 prior to normalisation. If the operator is immediately excluded again during the vector update process at the end of the readmission phase, the performance penalty, and hence the number of exclusion phases, is increased by one. Should the operator be readmitted and survive the vector update process into the succeeding phase, then the performance penalty is reset to one.

The AdaptiveHH used during the second stage of this thesis (see Chap-

ter 5) used only the Basic Selector [BS]. In this stage of this thesis we examine the relative performance of the 48 different pairings of operator selection vector and solution acceptance criteria described below and in Section 6.2. The operator selection vectors are of our own design, but use components of the single selection vector used by Misir et al. [64].

1. **[FS] Fixed Selector:** The initial vector is not altered during the run, so provides a benchmark against which other selection vectors can be measured. All operators have an equal probability of selection regardless of performance.

2. **[BS] Basic Selector:** Updates probabilities by evaluating the success rate of each operator, r_i, since the start of the run:

$$r_i = \frac{\text{number of improvements}_i}{\text{number of calls}_i}$$

   This vector does not exclude operators and sets a minimum probability of selection as 0.01 prior to normalisation.

3. **[P1] Phase Selector (1):** Updates probabilities by evaluating the success rate of the each operator, r_i, in the most recent phase:

$$r_i = \frac{\text{number of improvements}_i}{\text{number of calls}_i}$$

   During the update process a threshold is set equal to $\frac{1}{3}$ of the success rate of the best performing operator, r_{best}, in that phase. If $r_i \geq \frac{r_{best}}{3}$ it is included in the selection vector for the next phase with a probability of r_i, minimum 0.01, prior to normalisation. Operators where $r_i < \frac{r_{best}}{3}$ are excluded from the vector for the number of phases determined by their performance penalty.

4. **[P2] Phase Selector (2):** Updates probabilities by evaluating the success rate of the each operator, r_i, in the most recent phase:

$$r_i = \frac{\text{number of improvements}_i}{\text{number of calls}_i}$$

This vector does not exclude operators and sets a minimum probability of selection at 0.01 prior to normalisation. It differs from [BS] in that this selection vector only considers performance during the most recent phase.

5. **[T1] Time Weighted Phase Selector (1):** The time weighted selector uses a time weight, w_i, to penalise slower operators. This is calculated using the average operator execution time, $averageOpTime$, during the preceding phase:

$$w_i = \sqrt{\frac{averageOpTime_i}{averageOpTime_{fastest}}}$$

The time weighted success rate of each operator, r_i, in the most recent phase is evaluated:

$$r_i = \frac{\text{number of improvements}_i}{w_i \times \text{number of calls}_i}$$

During the update process a threshold is set equal to $\frac{1}{3}$ of r_{best}. If $r_i \geq \frac{r_{best}}{3}$ it is included in the selection vector for the next phase with a probability of r_i, minimum 0.01, prior to normalisation. Operators where $r_i < \frac{r_{best}}{3}$ are excluded from the vector for the number of phases determined by their performance penalty.

6. **[T2] Time Weighted Phase Selector (2):** Calculation of the time weight, w_i, and success rates, r_i, are identical to that described for [T1]. For this selector all r_i are ranked highest to lowest, including those excluded from the selection vector ($r_i = 0$). A threshold, T, is set equal to the r_i of the operator ranked $\frac{NumberOfOperators}{2}$ (T may be zero). If $r_i \geq T$, it is included in the selection vector for the next phase with a probability weighting of $\frac{1}{\text{rank}}$, prior to normalisation.

6.2 Acceptance Criteria Design

Each application of an operator takes a current solution and modifies it to create a new solution. The new solution is then considered for acceptance into the small population of solutions. If the new solution is not accepted then it is discarded. If the new solution is at least as good as the solution it will replace, then it is automatically accepted into the population of solutions regardless of the acceptance criteria specified by the hyper-heuristic. The following eight acceptance criteria are those proposed by Sabar et al. [80] with minor modifications. As far as possible we have retained the labels and arbitrary values proposed by Sabar et al. [80] and only made changes which are necessary to satisfy HyFlex framework [70] constraints. In all cases, the new solution is compared to the solution it will replace (if accepted) in the population of solutions.

1. **[IO] Improving or Equal Only:** Only improving (better objective value) or equally good solutions are accepted. All other solutions are discarded.

2. **[AM] Accept Move:** All new solutions are accepted.

3. **[SA] Simulated Annealing:** Non-improving solutions are accepted with a probability $e^{-\delta/t}$, where δ is the change in the objective value between the old and new solutions. The "temperature", t, is $0.5 \times S_{best} \times 0.85^{phase-1}$, where S_{best} is the current best solution objective value [83]. The probability of a non-improving solution being accepted decreases as (a) the change in objective value increases and (b) as time progresses.

4. **[MC] Exponential Monte Carlo:** Non-improving solutions are accepted with a probability $e^{-\delta}$, where δ is the change in the objective value between the old and new solutions. The probability of a non-improving solution being accepted decreases as the change in objective value, δ, increases.

5. **[RR] Record to Record Travel:** Non-improving solutions are accepted if the new solution has an objective value less than or equal to $1.03 \times S_{best}$, where S_{best} is the current best solution objective value [27].

6. **[GD] Great Deluge:** Non-improving solutions are accepted if the new solution has an objective value less than or equal to

$$(1 + 0.85^{phase-1}) \times S_{best}$$

where S_{best} is the current best solution objective value [27]. The probability of a non-improving solution being accepted decreases as time progresses.

7. **[NA] Naïve Acceptance:** Non-improving solutions are accepted with 0.5 probability.

8. **[AA] Adaptive Acceptance:** Non-improving solutions are accepted with a probability $1 - \frac{1}{C}$, where $C > 0$ is a counter which increments every 10,000 consecutive operator calls without an improvement in the objective value of the best solution found so far. The counter is reset to 1 each time an improved best solution objective value is found. The probability of a non-improving solution being accepted increases when the search for better solutions reaches a plateau and new best found solutions become harder to find.

6.3 Experimental Design

We test the effectiveness of AdaptiveHH2 by rating each solution generated against the best solution objective value achieved within the computational time limit. We use different pairings of operator selection vector and acceptance criterion. There are 48 possible pairings of operator selection vector (6) and solution acceptance criteria (8). Parameters also set the rules about how AdaptiveHH2 responds if progress towards improving the current solution is stalled.

These experiments use the CVRP domain described in Section 3.3 on page 36, which is compatible with the HyFlex [70] framework. We create 50 random 80-node (79 customers + 1 depot) problem instances requiring a minimum of between 3 and 19 routes. Each problem instance is randomly created using an 80×80 grid. Each instance contains three nodes at fixed locations (see Figure 6.2), one of which is the depot, and the other 77 nodes at randomly generated locations. Vehicle capacity is fixed at 1,000 units and each customer's demand is a randomly generated integer with an upper bound ranging from 5% to 45% (randomly set for each instance) of the vehicle capacity, with a minimum demand of 1 unit.

The size of the population of solutions is set at six. The hyper-heuristic is only provided with the number of operators of each operator type and has no knowledge of the actual function each operator performs.

We seek to determine:

1. Whether there are particular pairings of operator selection vector and acceptance criterion which consistently perform well or poorly compared to other pairings in arriving at a "good" solution within a short computational time. We examine how each pair affects the frequency with which each operator type is selected.

2. Whether the location of the depot in relation to the customers influences the consistency and quality of the solutions. To this end we take a problem instance (see Figure 6.2) and relocate the depot by swapping the grid coordinates of the depot with one of two customers highlighted. The problem instance is otherwise unchanged. The alternative depot locations are chosen so that the depot is geographically: (a) central, (b) off-centre, and (c) remote.

3. Although we use CVRP instances of the same size, the differing customer demand values mean solutions require a minimum number of routes ranging from 3 to 19. We examine the influence the number

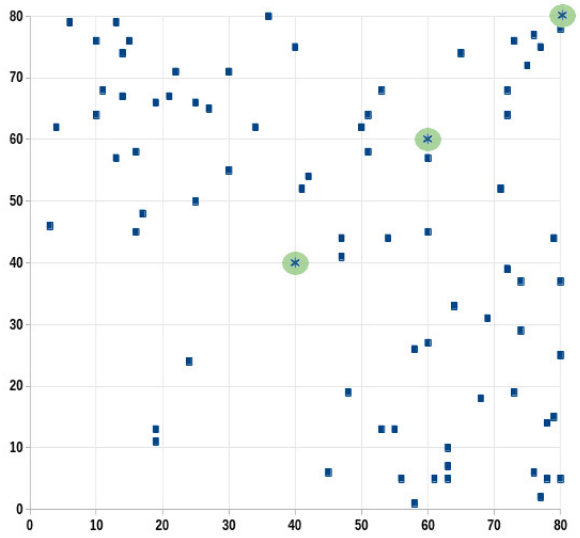

Figure 6.2: Randomly generated 80-node problem instance on an 80×80 grid, showing the 3 alternative depot locations (highlighted).

of routes has on the performance of the operator selection vector and acceptance criterion pairings.

We compare the quality of the results from 30 replications on a set of 50 randomly generated 80 node CVRP instances. We rate individual solutions against the best solution found during the batch of runs (typically 1,440 runs, being 30 replications of 48 pairings) using the following formula (lower ratings are better).

$$\text{rating}_i = \left(\frac{100 \times (\text{solution}_i - \text{solution}_{best})}{\text{solution}_{best}} \right)^2$$

This provides an indication of the relative performance of each pair compared to its peers.

We conduct 25-phase (see Section 5.1 on page 72) experiments using three different depot locations on 50 CVRP instances. We measure performance of each pair using computational time limits of 1, 5, 15, 30, 60, 120

and 300 seconds. AdaptiveHH2 includes a reinitialisation mechanism we have designed if no improving solutions are found for 10,000 consecutive operator calls (see Section 6.3.1). It also contains an early termination condition (see Section 6.3.1) should there still be no improvements to the best found solution for two consecutive phases. The purpose of this mechanism is to allow processing to be halted when the hyper-heuristic detects there is a very low likelihood of making further improvements to the best found solution so far.

6.3.1 Stalling

The best found solution may reach a state where no further improvements have been made for a considerable number of operator applications. We refer to this phenomenon as stalling. Assuming sufficient computational time is allowed, the rate at which the best found solution is improved will gradually diminish and eventually stall. This may, of course, be due to the best found solution being an optimal solution. Unfortunately, the hyper-heuristic has no means of knowing when an optimal solution has been reached. However, the lack of further improvements may also be due to solution development becoming stuck at a particular solution which is beyond the ability of the operators to break free from. This is commonly referred to as "local optima", but note that the local optima is only defined with respect to the operators available. The hyper-heuristic has no means of knowing the cause of the stalling, and must therefore apply a single set of actions in response.

For the hyper-heuristics conforming to the HyFlex [70] framework specifications we have followed the example of Misir et al. [64] and develop a multi-stage approach, repeating each stage multiple times before moving onto the next stage. The stages we have developed are:

1. **Global parameter adjustment**: The hyper-heuristic can instruct the problem domain to adjust the global α and β parameters (see Section

3.2 on page 33). This will adjust the performance of the operators that use either or both those parameters although the appropriate size of the required change is difficult to predict. This adjustment may be sufficient to enable an operator to create a new solution which directly or indirectly enables progress towards improving the current best found solution. In this thesis we set the threshold to trigger an adjustment in the global parameters every 500 consecutive operator calls without any improving solution.

2. **Reinitialise solutions**: If no further improvements to the best found solution occur from adjusting the global parameters, the next option is to reinitialise the population of solutions, other than the best found solution so far. This is a more drastic change which discards all but one of the current solutions. It can, however, provide the large scale change that may be able to overcome the cause of the stalling. Less drastic options include adding rather than replacing solutions in the population of solutions. In this thesis we set the threshold to trigger a reinitialisation at 10,000 consecutive operator calls without any improving solution.

3. **Change operator selection vector and/or solution acceptance criterion**: If both the preceding stages fail (or as an alternative to reinitialisation), then the operator selection vector and/or solution acceptance criterion can be replaced during an interim phase update. We omit this feature in AdaptiveHH2, but it is a recommended option in future research.

4. **Early termination**: If all these adjustment attempts fail, then the hyper-heuristic should consider terminating the computation and return the current best found solution.

   Since the hyper-heuristic has no detailed knowledge of the problem domain, nor of the problem instance size and features, there is no

means of determining the best computational time to allow. Consequently a manually specified computational time limit may be excessive or insufficient to arrive at a "good" solution. When faced with a previously unseen problem instance, it may be very difficult to manually estimate the appropriate computational time to allow. Consequently there is a tendency to err on the side of caution, and specify a generous computational time limit. However, in a commercial environment, a "good" solution may be required as soon as possible and unnecessary computational effort may be costly. We therefore include in the design of AdaptiveHH2 a mechanism that terminates the computation regardless of the computational time still available.

Setting the appropriate threshold to trigger early termination is difficult. This thesis focuses on appropriate criteria for the operators contained within the CVRP domain we have created. Further work is required to adjust these parameters and modify the criteria for a generic problem domain. In this thesis we enable early termination if either a target objective value (which may be set to zero) is reached or if no improvements to the objective value of the best found solution are found for N consecutive operator calls, P phases, or T time.

6.4 Results and Discussion

Tables 6.1 to 6.4 and Figure 6.3 show the results for all pairings from the batches using a 60 second computational time limit for each depot location. Table 6.5 shows results from the five best and five worst performing pairings identified in Tables 6.1 and 6.2. Widely differing customer demand values mean the 50 CVRP instances require a minimum of between 3 and 19 routes to service all customers. Tables 6.3 and 6.4 compare the performance of pairings on problem instances where the minimum number of routes is small (3–5 routes), medium (6–13 routes) and large (14–19 routes). Table 6.5 shows the change in performance over different compu-

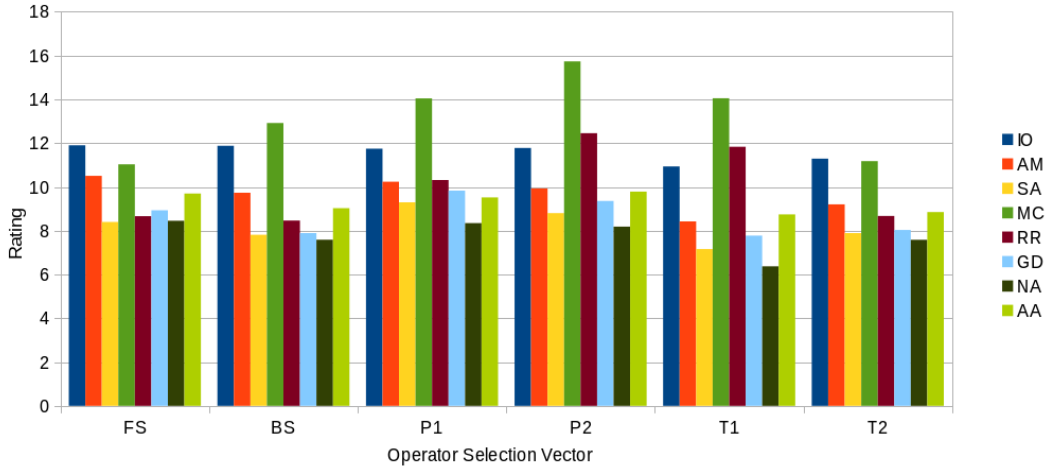

Figure 6.3: Comparison of acceptance criteria and selection vector performance during 60 seconds runs shown in Table 6.1. Lower ratings are better.

tational time limits. In Table 6.5 the performance is measured against the best solution found in any batch for each CVRP instance and depot location. Early terminations only affect the data when allowing a 300 seconds computational time limit. A negligible number (<0.1%) of early terminations occurred with a 120 second time limit, and none with the shorter time limits.

Tables 6.1 and 6.2 illustrate the difference in performance when the depot is at different locations. While all pairings provide better results when the depot is located centrally compared to off-centre, the better performing pairings generally show improving performance when the depot is moved even further away from the centre. In contrast, the poorer performing pairings generally show neutral to worsening results the farther the depot is located from the geographic centre. This highlights that the size of the problem instance is not the only factor influencing the performance of the hyper-heuristic.

Table 6.1: Part 1: Average rating of each selection vector and acceptance criteria pair over 3 depot locations × 30 replications × 60 seconds runs on 50 randomly generated CVRP instances (80 nodes, 3 depot locations (see Figure 6.2)). Lower ratings are better. Best five performing pairings in bold; five worst in italics. Also see Table 6.2.

Accept. Criteria	**Select. Vector**	**Central Depot**	**Off-cen. Depot**	**Remote Depot**	**Avge. Rating**	**Std. Dev.**
IO	FS	*10.33*	12.35	13.02	11.90	13.62
IO	BS	10.09	12.50	13.04	11.88	13.53
IO	P1	9.96	12.29	12.99	11.75	13.50
IO	P2	10.09	12.29	12.96	11.78	13.38
IO	T1	9.60	11.21	11.99	10.93	12.50
IO	T2	9.74	11.63	12.50	11.29	12.85
AM	FS	9.01	11.89	10.61	10.51	12.42
AM	BS	8.09	11.09	10.02	9.73	11.37
AM	P1	8.19	11.74	10.79	10.24	12.27
AM	P2	7.51	11.57	10.72	9.93	11.97
AM	T1	6.25	10.17	8.86	8.43	10.84
AM	T2	7.35	10.78	9.48	9.20	10.95
SA	FS	6.81	9.25	9.13	8.40	10.31
SA	BS	**5.93**	**8.56**	8.94	7.81	10.36
SA	P1	7.19	10.34	10.36	9.30	11.52
SA	P2	6.74	9.65	10.02	8.80	10.74
SA	T1	**5.59**	**8.06**	**7.83**	**7.16**	9.60
SA	T2	6.10	8.80	8.77	7.89	10.35
MC	FS	9.75	10.96	12.38	11.03	12.54
MC	BS	*11.02*	13.03	*14.70*	*12.92*	14.28
MC	P1	*11.92*	*14.34*	*15.87*	*14.04*	15.28
MC	P2	*13.49*	*16.35*	*17.36*	*15.73*	17.04
MC	T1	*11.77*	*15.01*	*15.37*	*14.05*	15.60
MC	T2	9.99	11.05	12.49	11.18	12.96

Table 6.2: Part 2: Average rating of each selection vector and acceptance criteria pair over 3 depot locations × 30 replications × 60 seconds runs on 50 randomly generated CVRP instances (80 nodes, 3 depot locations (see Figure 6.2)). Lower ratings are better. Best five performing pairings in bold; five worst in italics. Also see Table 6.1.

Accept. Criteria	Select. Vector	Central Depot	Off-cen. Depot	Remote Depot	Avge. Rating	Std. Dev.
RR	FS	6.77	9.46	9.76	8.66	10.88
RR	BS	6.50	9.25	9.63	8.46	10.76
RR	P1	7.73	11.30	11.91	10.31	12.27
RR	P2	9.21	*13.85*	*14.29*	*12.45*	14.21
RR	T1	9.29	*13.05*	13.16	11.83	13.55
RR	T2	6.55	9.30	10.17	8.68	11.01
GD	FS	7.66	9.33	9.81	8.93	11.14
GD	BS	6.47	**7.98**	9.22	7.89	10.29
GD	P1	8.15	10.35	11.00	9.83	11.92
GD	P2	7.57	9.82	10.69	9.36	11.31
GD	T1	6.15	8.74	**8.42**	**7.77**	10.32
GD	T2	6.72	**8.49**	8.88	8.03	10.33
NA	FS	6.71	9.66	8.99	8.45	10.61
NA	BS	**5.96**	8.68	**8.11**	**7.59**	10.01
NA	P1	6.38	9.63	9.04	8.35	10.51
NA	P2	6.33	9.35	8.86	8.18	10.20
NA	T1	**4.58**	**7.60**	**6.95**	**6.37**	8.88
NA	T2	**5.88**	8.64	**8.22**	**7.58**	9.84
AA	FS	8.58	10.60	9.92	9.70	11.58
AA	BS	8.08	9.48	9.51	9.03	10.85
AA	P1	8.28	10.31	9.96	9.52	11.55
AA	P2	8.12	10.74	10.49	9.78	11.91
AA	T1	7.04	9.93	9.26	8.75	11.29
AA	T2	7.62	9.55	9.39	8.85	11.04

Table 6.3: Comparison (part 1) of average rating of each selection vector and acceptance criteria pair on CVRP instances requiring a small (15 instances), medium (18 instances) and large (17 instances) minimum number of routes. Lower ratings are better. Best five performing pairings in bold; five worst in italics. Also see Table 6.4.

Acceptance Criteria	Selection Vector	3-5 routes	6-13 routes	14-19 routes	Average Rating	Std.Dev.
IO	FS	*8.58*	12.58	13.64	11.90	13.62
IO	BS	8.20	12.52	*13.93*	11.88	13.53
IO	P1	8.30	12.56	13.43	11.75	13.50
IO	P2	8.04	12.39	13.92	11.78	13.38
IO	T1	7.57	11.60	12.73	10.93	12.50
IO	T2	8.18	11.79	13.07	11.29	12.85
AM	FS	6.13	12.42	11.60	10.51	12.42
AM	BS	5.65	11.45	10.84	9.73	11.37
AM	P1	5.88	12.00	11.49	10.24	12.27
AM	P2	6.37	12.12	10.07	9.93	11.97
AM	T1	6.23	10.56	**7.59**	8.43	10.84
AM	T2	**5.40**	10.79	10.25	9.20	10.95
SA	FS	5.91	9.00	9.60	8.40	10.31
SA	BS	5.67	**8.59**	8.54	7.81	10.36
SA	P1	6.44	10.02	10.63	9.30	11.52
SA	P2	6.16	9.38	10.15	8.80	10.74
SA	T1	6.30	**8.25**	**6.53**	**7.16**	9.60
SA	T2	5.49	**8.67**	8.81	7.89	10.35
MC	FS	7.65	11.84	12.66	11.03	12.54
MC	BS	8.24	*14.16*	*15.03*	*12.92*	14.28
MC	P1	*9.56*	*15.27*	*16.03*	*14.04*	15.28
MC	P2	*11.55*	*16.90*	*17.56*	*15.73*	17.04
MC	T1	*11.84*	*15.99*	13.45	*14.05*	15.60
MC	T2	7.78	11.97	12.84	11.18	12.96

Table 6.4: Comparison (part 2) of average rating of each selection vector and acceptance criteria pair on CVRP instances requiring a small (15 instances), medium (18 instances) and large (17 instances) minimum number of routes. Lower ratings are better. Best five performing pairings in bold; five worst in italics. Also see Table 6.3.

Acceptance Criteria	Selection Vector	3-5 routes	6-13 routes	14-19 routes	Average Rating	Std.Dev.
RR	FS	6.57	9.09	9.76	8.66	10.88
RR	BS	5.76	8.84	10.09	8.46	10.76
RR	P1	6.94	10.65	12.50	10.31	12.27
RR	P2	7.94	*12.97*	*15.29*	*12.45*	14.21
RR	T1	*9.51*	12.80	12.47	11.83	13.55
RR	T2	6.05	8.87	10.46	8.68	11.01
GD	FS	5.78	10.03	10.05	8.93	11.14
GD	BS	**5.18**	8.83	8.86	7.89	10.29
GD	P1	5.83	10.96	11.57	9.83	11.92
GD	P2	6.40	10.20	10.63	9.36	11.31
GD	T1	6.41	9.01	**7.35**	**7.77**	10.32
GD	T2	**5.40**	8.77	9.18	8.03	10.33
NA	FS	5.76	9.82	8.89	8.45	10.61
NA	BS	**5.30**	**8.71**	8.01	**7.59**	10.01
NA	P1	5.93	9.55	8.78	8.35	10.51
NA	P2	6.57	8.91	8.56	8.18	10.20
NA	T1	5.84	**7.43**	**5.54**	**6.37**	8.88
NA	T2	**5.30**	8.77	**7.93**	**7.58**	9.84
AA	FS	6.32	11.08	10.66	9.70	11.58
AA	BS	5.94	9.94	10.31	9.03	10.85
AA	P1	6.56	10.53	10.60	9.52	11.55
AA	P2	6.72	10.99	10.71	9.78	11.91
AA	T1	6.60	10.14	8.75	8.75	11.29
AA	T2	6.14	9.74	9.88	8.85	11.04

As shown in Tables 6.1 and 6.2, some pairings, such as [SA][T1] and [NA][T1], consistently perform better than other pairings. The pairings using the [MC] and [IO] acceptance criteria generally perform poorly and require a longer computational time to achieve results competitive with the better performing pairings

A possible cause of this difference is the diversity in the population of solutions. Pairings using the [IO] acceptance criterion, and to a lesser extent [MC], work with a smaller diversity of interim solutions compared to other forms of acceptance criteria. This means that time and effort are not lost on improving low quality solutions that may never become the best solution in the current population of interim solutions. This is a useful trait if the computational time limit is very short, since effort is directed towards improving a better quality solution. On the other hand, accepting only improving or equally good solutions can cause the population of solutions to stagnate and eventually become clones of the best found solution. Once this stage is reached the crossover operators become ineffective and there is a tendency for the process to stall. The hyper-heuristic has a mechanism to reinitialise the population of solutions in the event of stalling, but this is only effective if the selection vector and acceptance criterion pairing can avoid regenerating the same set of solutions.

Tables 6.1 and 6.2 also confirm that there is an inter-dependency between the operator selection vector and the acceptance criterion and it is insufficient to separately evaluate each, even though they carry out different functions. An increase in the number of routes (see Tables 6.3 and 6.4) as well as the relative location of the depot are influencing factors as well. However, Tables 6.1 to 6.4 also show that the **relative** performance of operator selection vector and acceptance criterion pairings compared to other pairings is not greatly altered by the number of routes or depot location. A better performing pairing will consistently deliver higher quality solutions than poorer performing pairings regardless of the depot location or the minimum number of routes.

Table 6.5: Average rating of five best and five worst performing pairings from Tables 6.1 and 6.2 on 50 CVRP instances $\times$ 3 depot locations $\times$ 30 replications, when allowing a different computational time limit (in seconds). All ratings are measured against the best solution to each instance (and depot location) found during any of the seven batches.

Accept.	**Select.**	**1sec.**	**5sec.**	**15s**	**30s**	**60s**	**120s**	**300s**
NA	T1	45.27	18.85	12.39	9.25	6.88	5.37	4.07
SA	T1	48.08	20.47	14.14	10.52	7.72	5.88	4.28
NA	T2	888.03	22.97	13.96	10.81	8.19	6.34	4.67
NA	BS	57.72	19.90	13.79	10.89	8.19	6.16	4.49
GD	T1	47.08	22.33	14.97	11.27	10.55	6.40	4.63
RR	P2	46.62	25.11	18.92	15.88	13.29	10.72	8.16
MC	BS	60.19	23.67	18.64	16.06	13.73	11.77	9.69
MC	P1	52.42	29.00	21.62	18.00	14.85	12.35	9.93
MC	T1	54.62	28.95	21.95	18.82	14.88	12.03	9.83
MC	P2	51.74	30.51	23.19	20.01	16.62	13.53	10.96

Table 6.5 shows the performance of each pair improves with a longer computational time limit, but not all improve at the same rate. The [NA][T2] pair performs poorly with the 1 second computational time limit but well with longer time limits, indicating a minimum time limit (or number of operator calls) per phase is necessary for some pairings before the operator selection vector update process can be effective. In these experiments the improvement in the performance of the better performing pairings appears to be reaching a plateau with a 300 second time limit. However, the poorer performing pairings show a non-trivial improvement in performance between 120 and 300 seconds time limits, suggesting a longer computational time may produce further improvements.

Table 6.6 illustrates how different pairings of operator selection vector

Table 6.6: Average number of operator calls, success rate (r_i, as defined in Section 6.1) and mix of operator type selection between the six selection vectors (SV) and the Naïve Acceptance [NA], Exponential Monte Carlo [MC] and Improving or Equal Only [IO] acceptance criteria (AC) during experiments using a 60 seconds computational time limit. Best performing pairings (from Tables 6.1 and 6.2) in bold, worst in italics.

AC	SV	Number calls	Success Rate	Mutation	Ruin-Recreate	Local Search	Crossover
NA	FS	236,900	11.82%	33.34%	16.66%	33.33%	16.67%
NA	**BS**	**168,900**	**8.12%**	**12.65%**	**15.44%**	**47.01%**	**24.89%**
NA	P1	138,300	5.53%	5.86%	10.41%	54.29%	29.44%
NA	P2	231,000	4.71%	4.40%	3.57%	29.40%	62.63%
NA	**T1**	**749,400**	**5.78%**	**2.34%**	**2.88%**	**3.53%**	**91.24%**
NA	**T2**	**187,500**	**6.82%**	**17.08%**	**14.81%**	**45.22%**	**22.89%**
MC	FS	220,800	1.43%	33.36%	16.64%	33.34%	16.66%
MC	*BS*	*253,600*	*2.38%*	*7.11%*	*5.71%*	*30.27%*	*56.91%*
MC	*P1*	*246,100*	*2.54%*	*3.41%*	*2.61%*	*30.81%*	*63.17%*
MC	*P2*	*269,600*	*2.70%*	*3.30%*	*1.64%*	*23.59%*	*71.47%*
MC	*T1*	*821,400*	*2.73%*	*1.68%*	*0.84%*	*1.68%*	*95.80%*
MC	T2	323,800	1.79%	51.81%	11.63%	22.07%	14.49%
IO	FS	229,900	0.16%	33.32%	16.66%	33.35%	16.67%
IO	BS	196,100	0.14%	18.31%	18.28%	39.00%	24.41%
IO	P1	201,000	0.15%	25.29%	18.55%	35.22%	20.94%
IO	P2	191,800	0.16%	22.32%	18.20%	33.23%	26.25%
IO	T1	512,100	0.18%	34.95%	8.69%	8.55%	47.81%
IO	T2	373,200	0.16%	56.22%	12.51%	18.85%	12.41%

and acceptance criterion affect the frequency with which particular operator types are called. The number of calls illustrates how the more aggressive of the two time weighted selection vectors, [T1], biases operator selection towards the faster mutation and crossover operators and away from the slower local search operators. The second time weighted selection vector, [T2], maintains are more balanced selection approach. The Fixed Selector [FS] reflects the 4:2:4:2 balance between the four operator types in the CVRP domain.

The time-weighted selectors favour the faster mutation operators at the expense of the slower search operators. This is a trade-off between speed and quality. Table 6.6 shows that a large number of operator calls is not critical to the quality of the solution. This table also illustrates the lower number of calls made to crossover type operators when the [IO] acceptance criteria is used, reflecting the reduced effectiveness of these operators in this situation. In contrast the time weighted selector [T1] almost exclusively uses the crossover operator with both the Naïve Acceptance [NA] and Exponential Monte Carlo [MC] acceptance criteria. With [NA], the resulting solutions are among the best, while with [MC] they are among the worst. This can be explained by the difference in the diversity of the population of solutions, as reflected in the relative operator call success rates. However other factors such as parameter values and the number of early terminations (42% during 300 seconds time limit) due to best found solutions no longer improving, may also influence the difference in overall solution quality.

6.5 Stage 3 Conclusions

These experiments show that the performance of an adaptive hyper-heuristic is influenced by the choice of operator selection vector and solution acceptance criterion pairing. However, the *relative* performance of each pairing is not greatly changed by different features of the problem instance. As

noted by Sabar et al. [80], maintaining a small population of solutions provides a greater diversity of solutions. This in turn reduces the likelihood of stalling, and so minimises the adverse effects of drastic changes such as reinitialisation. We have used a number of manually set parameter settings, particularly in relation to the mechanisms we have designed to handle stalling and early termination of the computation. While the chosen parameters appear to work satisfactorily, there is scope for further research into this aspect of the AdaptiveHH2 design.

When comparing the results in Tables 6.1 to 6.5 we deduce the following about the operator selection vector and acceptance criteria pairings.

1. Generally perform well:

   (a) Time Weighted Phase Selectors [T1] and [T2] with the Naïve Acceptance [NA] criterion. The time weighted operator selection vectors bias selection of operators towards the faster operators. Slower operators are either excluded (with [T1]) or the probability of selection greatly reduced (with [T2]). Consequently relatively few operators are employed during a single phase. In contrast, the [NA] acceptance criterion accepts more non-improving solutions than most other acceptance criteria. This combination of concentrated application of a few fast operators on a moderately diverse set of solutions seems to work well.

   (b) Time Weighted Phase Selector (1) [T1] with Simulated Annealing [SA] acceptance criterion. This combination allows more non-improving solutions to be admitted to the population in the early stages of computation, but progressively moves towards accepting only improving or equally as good solutions.

2. Generally perform poorly:

   (a) Any operator selection vector with the Exponential Monte Carlo acceptance criterion [MC]. Further research is required to fully

understand why pairings using the [MC] acceptance criterion consistently perform poorly. Preliminary investigations indicate the lack of diversity in the population of solutions hinders the ability of the preferred operators to work effectively.

(b) Any operator selection vector with the Improving or Equal Only acceptance criterion [IO]. The lack of diversity in the population of solutions is the most likely cause of the poorer performance of pairings using the [IO] acceptance criterion. Since the [IO] acceptance criterion was used in stage one of this thesis (see Chapter 5), the performance of both of the hyper-heuristics may improve further, in absolute terms, if a different acceptance criterion is used.

Further research is recommended in the following areas:

1. Enabling the hyper-heuristic to change the choice of operator selection vector and/or solution acceptance criterion at an interim phase update.

2. Investigating performance using problem instances of different sizes and in different domains.

3. Determining a good number of phases, and whether each phase should be limited by time or by number of operators calls (or a combination of both).

4. The design and parameters for the mechanisms to handle stalling and early terminations.

Summary

In this chapter we have investigated different pairings of operator selection vector and solution acceptance criteria and established that some pairings perform better than others. We also show that the number of operator

calls does not influence the solution quality, indicating quality over quantity generally leads to better outcomes. In the next chapter we summarise the research done in this thesis and suggest areas for further research.

Chapter 7

Conclusions

The goal of this thesis, outlined in Section 1.2 on page 4, is to extend current hyper-heuristic research towards answering the question: How can a hyper-heuristic efficiently and effectively adapt the selection, generation and manipulation of domain specific heuristics as you move from small size and/or narrow domain problems to larger size and/or wider domain problems, i.e., managing *scalability*?

The subject is too large to adequately cover in a single Masters thesis, so, as detailed in Section 1.3 on page 5, we have investigated three aspects of hyper-heuristic scalability:

1. In the first stage of this thesis we investigate whether a single heuristic can handle scalability issues, or if different heuristics are required for different sized problems. We conclude that it is possible for a hyper-heuristic to evolve a heuristic for a specific problem domain that can deliver "good" solutions to problem instances of different sizes and containing different features. While the data structures and design of the CVRP domain used in the first stage of this thesis limit the effectiveness of the local search heuristic on larger size problem instances, we conclude that this is not as a result of the hyper-heuristic design. The deterministic local search we developed is effective, but its implementation in the CVRP domain is inefficient

and the computational time becomes excessive on large sized problem instances.

2. In the second stage of this thesis we establish whether some types of hyper-heuristic respond to scalability issues better or worse than other types of hyper-heuristic. We conclude that hyper-heuristics do not handle scalability issues equally well. The flexibility of the adaptive hyper-heuristic enables better performance than the more rigid structure of the grammar based hyper-heuristic, even though the grammar we use generates a heuristic similar in structure to the successful heuristics created in stage one.

   We also note that the grammar-based hyper-heuristic using off-line learning provides a compact and easily understood heuristic, which is useful if a reusable heuristic is required. The adaptive hyper-heuristic dynamically customises the development of a solution for a given problem instance, so is closer in approach to the grammar-based hyper-heuristic using on-line learning. We also conclude that both hyper-heuristics used during the second stage of this thesis can be readily applied to different problem domains.

3. In the third stage of this thesis we investigate how the adaptive hyper-heuristic developed in the second stage responds to problem instances of the same size, but containing different features and complexity. We also apply different computational budgets to monitor the effect of the computational time limit on scalability issues. During this stage we identify which of 48 possible pairings of the key components used by the adaptive hyper-heuristic perform well, and which perform poorly. We enhance the adaptive hyper-heuristic to enable a wider choice of operator selection vector (SV) and solution acceptance criteria (AC). Analysis of quality of solutions obtained when using different pairings of SV and AC (Tables 6.1 and 6.2 on pages 98 and 99) show some pairings perform better than others. However,

as shown with the [NA][T2] AC:SV pairing in Table 6.5 on page 103, an AC:SV pairing which generally performs well can produce very poor results if the computational time limit, or one of the parameters, is not set at a good value.

7.1 Contributions

This thesis contributes to the general understanding of scalability issues when using hyper-heuristics. Each stage of this thesis contributes as follows:

1. **Stage 1** differs from previous research (see Section 2.3 on page 25) in that we only use *deterministic* low-level operators to manipulate solutions to CVRP instances. We show that competitive results can be achieved in this way.

2. **Stage 2** contributes by building and comparing two hyper-heuristics of contrasting design, and learning some of their respective strengths and weaknesses across seven different problem domains. A comparative performance analysis of this nature has not been previously researched. We show that the adaptive hyper-heuristic we developed can consistently deliver better results than the grammar-based hyper-heuristic.

3. **Stage 3** shows that scalability issues can arise from different features of the problem instance and the computational budget, and that instance size is not the only cause of scalability issues. This thesis contributes by learning how 48 different pairings of operator selection vector and solution acceptance criteria perform with different computational budgets.

7.2 Recommendations for Further Research

As stated at the beginning of this chapter, the subject of scalability is very large and completion of this thesis opens further questions for future research. Some recommendations for future research arise as a result of the work done for this thesis:

1. **Deterministic Local Search Operator**: If the implementation of the local search operator can be modified to streamline (a) identification of nearest neighbours and (b) evaluation of swaps, then this search operator may be very effective. Implementation of this search as an option within the CVRP domain operators described in Section 3.3.1 on page 37 would enable better evaluation of the value of this search operator.

2. **Validation of Operator Selection Vector and Solution Acceptance Criteria Pairings**: Further experiments using different problem instance sizes and different problem domains would establish whether the better performing pairings identified in the stage three experiments continue to perform well, or whether other pairings can before better in certain circumstances.

3. **Computational Time Limit**: In this thesis we manually specified the maximum computational time limit. If the hyper-heuristic is to be truly independent of the low-level problem domain and instance, then the hyper-heuristic needs to be able to determine an appropriate computational time and not rely on a manually set parameter. This is an area which would require considerable further research to be effective. We have provided an early termination mechanism in AdaptiveHH2, but further work is required on this to make the mechanism suitable for generic problem domains. The early termination parameters we have set are relevant to the CVRP domain we have used, but may be too large or too small for other domains.

4. **Changing Selection Vector and Acceptance Criteria**: As suggested in Section 6.3.1 on page 94, and illustrated in Table 6.5 on page 103, an adaptive hyper-heuristic may be able to avoid some performance issues if it were allowed to change the operator selection vector and solution acceptance criteria pair during a phase update process. Doing this may be a more effective way of overcoming solution development stalling than simply reinitialising the solutions.

5. **Grammar-guided Parameter Setting**: While we have shown that a hyper-heuristic can handle scalability issues, there remain a large number of manually and arbitrarily set parameters. Future research could establish which of these parameters are critical to the quality of the outcome. Can these parameters be set using an automated method? Although we have identified an adaptive hyper-heuristic performs better than one developed using grammar based genetic programming (GGGP), there is scope for a GGGP approach to be taken to set the various parameters for the adaptive hyper-heuristic. Setting and adjusting the HyFlex [70] global parameters, α and β, is particularly problematic, and there is scope for a GGGP approach to be taken towards defining rules to initialise and update these global parameters.

6. **Time-weighted Operator Selection Vectors**: Further research into the performance of the time-weighted operator selection vectors (see Section 6.1 on page 86) on problem instances of different sizes would enable appropriate adjustments to be made to the arbitrary thresholds set with these vector types. Again, this may be a situation where a GGGP approach may help adjust or design new operator selection vectors.

7. **Recreating and Improving Established (Meta-)heuristics**: Can a grammar-based hyper-heuristic evolve or describe an established meta-heuristic from component parts? For example, can a hyper-heuristic recreate

Tabu Search [34], Iterated Local Search [48] or Simulated Annealing [83]?

8. **Identifying Scalability Issues**: Since the hyper-heuristic has no knowledge of the problem instance size or features, how can the hyper-heuristic identify and adapt to instances with different features?

9. **Domain Memory**: A hyper-heuristic may be required to repeatedly develop heuristics to solve almost identical problem instances. How can the hyper-heuristic use prior knowledge to help deliver a "good" heuristic in the shortest time possible? How can the trade-off between the value of knowledge and the cost of storing and retrieving information be managed?

10. **Transfer Learning**: Can a heuristic developed for one problem domain be successfully applied to problem instances in another domain, e.g., VRP $\rightarrow$ VRP with pick-up and delivery [73]?

Bibliography

[1] *Capacitated Vehicle Routing Problem Instances*. http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/, October 2013.

[2] ALTINKEMER, K., AND GAVISH, B. Parallel savings based heuristic for the delivery problem. *Operations Research* (1991), 456–469.

[3] ARGELICH, J., AND MANYÁ, F. Exact MAX-SAT solvers for over-constrained problems. *Journal of Heuristics 12*, 4 (2006), 375–392.

[4] AUGERAT, P., BELENGUER, J. M., BENAVENT, E., CORBÉRAN, A., AND NADDEF, D. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research 106*, 2 (1998), 546–557.

[5] BADER-EL-DEN, M., AND POLI, R. Grammar-based genetic programming for timetabling. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'09)* (2009), pp. 2532–2539.

[6] BADER-EL-DEN, M., POLI, R., AND FATIMA, S. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing 1* (2009), 205–219.

[7] BAI, R., BURKE, E., KENDALL, G., LI, J., AND MCCOLLUM, B. A hybrid evolutionary approach to the nurse rostering problem. *IEEE Transactions on Evolutionary Computation 14*, 4 (2010), 580–590.

[8] BLUM, C., AND ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys 35*, 3 (2003), 268–308.

[9] BRÄYSY, O., AND GENDREAU, M. Vehicle routing problems with time windows, part I: Route construction and local search algorithms. *Transportation Science 39* (February 2005), 104–118.

[10] BURKE, E., GENDREAU, M., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND QU, R. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society 64*, 12 (2013), 1695–1724.

[11] BURKE, E., HYDE, M., AND KENDALL, G. Providing a memory mechanism to enhance the evolutionary design of heuristics. In *Proceedings of the IEEE World Congress on Computational Intelligence* (July 2010), pp. 3883–3890.

[12] BURKE, E., HYDE, M., AND KENDALL, G. Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation 16*, 3 (2012), 406–417.

[13] BURKE, E., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. Exploring hyper-heuristic methodologies with genetic programming. In *Computational Intelligence: Collaboration, Fusion and Emergence*, C. Mumford and L. Jain, Eds. Springer, 2009, pp. 177–201.

[14] BURKE, E., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds., vol. 146. Springer, 2010, pp. 449–468.

[15] BURKE, E., HYDE, M., KENDALL, G., AND WOODWARD, J. A genetic programming hyper-heuristic approach for evolving two di-

mensional strip packing heuristics. *IEEE Transactions on Evolutionary Computation 14*, 6 (2010), 942–958.

[16] BURKE, E., KENDALL, G., AND SOUBEIGA, E. A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics 9*, 3 (2003), 451–470.

[17] CLARKE, G., AND WRIGHT, J. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research 12*, 4 (1964), 568–581.

[18] COOK, W., CUNNINGHAM, W. H., PULLEYBLANK, W. R., AND SCHRIJVER, A. *Combinatorial Optimization*. Wiley, 1997.

[19] CORDEAU, J.-F., GENDREAU, M., LAPORTE, G., POTVIN, J.-Y., AND SEMET, F. A guide to vehicle routing heuristics. *The Journal of the Operational Research Society 53*, 5 (2002), 512–522.

[20] CORDEAU, J.-F., LAPORTE, G., AND MERCIER, A. A unified tabu search heuristic for vehicle routing problems with time windows. *The Journal of the Operational Research Society 52*, 8 (2001), 928–936.

[21] COUCHET, J., MANRIQUE, D., RÍOS, J., AND RODRÍGUEZ-PATÓN, A. Crossover and mutation operators for grammar-guided genetic programming. *Soft Computing 11*, 10 (2007), 943–955.

[22] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A hyper-heuristic approach for scheduling a sales summit. In *Selected Papers of the Third International Conference on the Practice and Theory of Automated Timetabling (PATAT)* (2000), vol. 2079 of *Lecture Notes in Computer Science*, Springer, pp. 176–190.

[23] CROES, G. A method for solving traveling salesman problems. *Operations Research 6* (1958), 791–812.

[24] DANTZIG, G., AND RAMSER, J. The truck dispatching problem. *Management Science 6* (1959), 80–91.

[25] DESROCHERS, M., AND VERHOOG, T. A matching based savings algorithm for the vehicle routing problem. Tech. rep., Les Cahiers du GERAD G-89-04, Montréal, Canada, 1989.

[26] DRAKE, J., KILILIS, N., AND ÖZCAN, E. Generation of VNS components with grammatical evolution for vehicle routing. In *Proceedings of the 16th European Conference on Genetic Programming* (2013), vol. 7831 of *Lecture Notes in Computer Science*, Springer, pp. 25–36.

[27] DUECK, G. New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational Physics 104*, 1 (1993), 86–92.

[28] EILON, S., WATSON-GANDY, C. D. T., AND CHRISTOFIDES, N. *Distribution Management: Mathematical Modelling and Practical Analysis*. Griffin, London, 1971.

[29] FISHER, M. Optimal solution of vehicle routing problems using minimum k-trees. *Operations Research 42*, 4 (1994), 626–642.

[30] FISHER, M., AND JAIKUMAR, R. A generalized assignment heuristic for vehicle routing. *Networks 11* (1981), 109–124.

[31] GAREY, M., AND JOHNSON, D. S. *Computers and Intractability*. W.H. Freeman, 2002.

[32] GENDREAU, M., LAPORTE, G., AND POTVIN, J.-Y. Metaheuristics for the vehicle routing problem. Tech. rep., Les Cahiers du GERAD G-98-52 (revised), Montréal, Canada, 1999.

[33] GILLET, B., AND MILLER, L. A heuristic algorithm for the vehicle dispatch problem. *Operations Research 22* (1974), 340–349.

[34] GLOVER, F. Tabu search: Part I. *ORSA Journal on Computing 1*, 3 (1989), 190–206.

[35] GLOVER, F. Tabu search: Part II. *ORSA Journal on Computing 2*, 1 (1990), 4–32.

[36] GOEL, A., AND GRUHN, V. *A General Vehicle Routing Problem*. Elsevier Science, Germany, 2006.

[37] HARPER, R., AND BLAIR, A. A structure preserving crossover in grammatical evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2005), vol. 3, pp. 2537–2544.

[38] JOSHI, A., AND SCHABES, Y. Tree-adjoining grammars. *Handbook of Formal Languages, Beyond Words 3* (1997), 69–123.

[39] KELLER, R., AND POLI, R. Self-adaptive hyperheuristic and greedy search. In *Proceedings of the IEEE World Congress on Computational Intelligence* (June 2008), pp. 3801–3808.

[40] KELLER, R., AND POLI, R. Subheuristic search and scalability in a hyperheuristic. In *Proceedings of the 10th Annual Genetic and Evolutionary Computation Conference (GECCO)* (2008), pp. 609–610.

[41] KELLER, R., AND POLI, R. Toward subheuristic search. In *Proceedings of the IEEE World Congress on Computational Intelligence* (June 2008), pp. 3148–3155.

[42] KENDALL, G., AND LI, J. Competitive travelling salesman problem: A hyper-heuristic approach. *Journal of the Operational Research Society 64*, 2 (2013), 208–216.

[43] KINDERWATER, G., AND SAVELSBERGH, M. Vehicle routing: Handling edge exchanges. In *Local Search in Combinatorial Optimization* (Chichester, England, 1997), E. Aarts and J. Lenstra, Eds., Wiley.

[44] KOZA, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[45] LAPORTE, G. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research 59* (1992), 345–358.

[46] LENSTRA, J., AND RINNOOY KAN, A. Complexity of vehicle routing and scheduling problems. *Networks 11* (1981), 221–227.

[47] LIN, S., AND KERNIGHAN, B. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research 21*, 2 (Mar - Apr 1973), 498–516.

[48] LOURENÇO, H., MARTIN, O., AND STÜTZLE, T. Iterated local search. *Handbook of Metaheuristics* (2003), 321–354.

[49] LUKE, S. *Essentials of Metaheuristics*, second ed. Lulu, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[50] LUKE, S., PANAIT, L., BALAN, G., PAUS, S., SKOLICKI, Z., KICINGER, R., POPOVICI, E., SULLIVAN, K., HARRISON, J., BASSETT, J., HUBLEY, R., DESAI, A., CHIRCOP, A., COMPTON, J., HADDON, W., DONNELLY, S., JAMIL, B., ZELIBOR, J., KANGAS, E., ABIDI, F., MOOERS, H., O'BEIRNE, J., TALUKDER, K. A., AND MCDERMOTT, J. *Evolutionary Computation in Java*. http://cs.gmu.edu/∼eclab/projects/ecj/, May 2014.

[51] MANRIQUE, D. *Artificial Neural Networks Design and New Optimisation Techniques Using Genetic Algorithms*. PhD thesis, Universidad Politécnica de Madrid, 2001.

[52] MANRIQUE, D., RÍOS, J., AND RODRÍGUEZ-PATÓN, A. Grammar-guided genetic programming. In *Encyclopedia of Artificial Intelligence*

(2008), J. Rabuñal, J. Dorado, and A. Pazos, Eds., Information Science Reference, pp. 767–773.

[53] MARSHALL, R., JOHNSTON, M., AND ZHANG, M. A comparison between two evolutionary hyper-heuristics for combinatorial optimisation. In *Proceedings of the 10th International Conference on Simulated Evolution and Learning (SEAL)* (2014), vol. 8886 of *Lecture Notes in Computer Science*, Springer, pp. 618–630.

[54] MARSHALL, R., JOHNSTON, M., AND ZHANG, M. Developing a hyper-heuristic using grammatical evolution and the capacitated vehicle routing problem. In *Proceedings of the 10th International Conference on Simulated Evolution and Learning (SEAL)* (2014), vol. 8886 of *Lecture Notes in Computer Science*, Springer, pp. 668–679.

[55] MARSHALL, R., JOHNSTON, M., AND ZHANG, M. Hyper-heuristics, grammatical evolution and the capacitated vehicle routing problem. *unpublished* (January 2014).

[56] MARTELLO, S., AND TOTH, P. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, 1990.

[57] MCKAY, R., HOAI, N., WHIGHAM, P., SHAN, Y., AND O'NEILL, M. Grammar-based genetic programming: A survey. *Genetic Programming and Evolvable Machines 11*, 3–4 (2010), 365–396.

[58] MCKAY, R., HOANG, T., ESSAM, D., AND NGUYEN, X. Developmental evaluation in genetic programming: the preliminary results. In *Proceedings of the 9th European Conference on Genetic Programming* (2006), P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, Eds., vol. 3905 of *Lecture Notes in Computer Science*, Springer, pp. 280–289.

[59] MEI, Y., LI, X., AND YAO, X. Co-operative co-evolution with route distance grouping for large-scale capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation 18*, 3 (2014), 435–449.

[60] MEI, Y., TANG, K., AND YAO, X. Decomposition-based memetic algorithm for multi-objective capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation 15*, 2 (2011), 151–165.

[61] MISIR, M., VERBEECK, K., DE CAUSMAECKER, P., AND VANDEN BERGHE, G. Design and analysis of an evolutionary selection hyper-heuristic framework. Tech. rep., CODeS Research Group, KAHO Sint-Lieven, K.U.Leuven Campus Kortrijk, Belgium, http://allserv.kahosl.be/∼mustafa.misir/papers/misir2012designTechRp.pdf, 2012.

[62] MISIR, M., VERBEECK, K., DE CAUSMAECKER, P., AND VANDEN BERGHE, G. The effect of the set of low-level heuristics on the performance of selection hyper-heuristics. In *Parallel Problem Solving from Nature - PPSN XII*, C. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, Eds., vol. 7492 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 408–417.

[63] MISIR, M., VERBEECK, K., DE CAUSMAECKER, P., AND VANDEN BERGHE, G. An investigation on the generality level of selection hyper-heuristics under different empirical conditions. *Applied Soft Computing 13*, 7 (July 2013), 3335–3353.

[64] MISIR, M., VERBEECK, K., DE CAUSMAECKER, P., AND VANDEN BERGHE, G. A new hyper-heuristic as a general problem solver: an implementation in HyFlex. *Journal of Scheduling 16* (2013), 291–311.

[65] MURPHY, E., O'NEILL, M., GALAVÁN-LOPÉZ, E., AND BRABAZON, A. Tree-adjunct grammatical evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2010), pp. 1–8.

[66] NAGATA, Y., AND BRÄYSY, O. Edge assembly based memetic algorithm for the capacitated vehicle routing problem. *Networks 54*, 4 (2009), 205–215.

[67] NAUR, P. Revised report on the algorithmic language ALGOL 60. In *Communications of the ACM* (1960), vol. 3, pp. 299–314.

[68] NEUMANN, J. V. Zur theorie der gesellschaftsspiele (on the theory of games of strategy). *Mathematische Annalen 100*, 1 (1928), 295–320.

[69] OCHOA, G., AND HYDE, M. *Cross-domain Heuristic Search Challenge*. http://www.asap.cs.nott.ac.uk/external/chesc2011/, 2011.

[70] OCHOA, G., HYDE, M., CURTOIS, T., VAZQUEZ-RODRIGUEZ, J., WALKER, J., GENDREAU, M., KENDALL, G., MCCOLLUM, B., PARKES, A., PETROVIC, S., AND BURKE, E. Hyflex: A benchmark framework for cross-domain heuristic search. In *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP)* (2012), vol. 7245 of *Lecture Notes in Computer Science*, pp. 136–147.

[71] OCHOA, G., QU, R., AND BURKE, E. Analysing the landscape of a graph based hyper-heuristic for timetabling problems. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2009), pp. 341–348.

[72] O'NEILL, M., HEMBERG, E., GILLIGAN, C., BARTLEY, E., MCDERMOTT, J., AND BRABAZON, A. *GEVA = Grammatical Evolution in Java*. http://ncra.ucd.ie/geva/, 2011.

[73] PISINGER, D., AND ROPKE, S. A general heuristic for vehicle routing problems. *Computers & Operations Research 34* (2007), 2403–2435.

[74] POLI, R., LANGDON, W. B., AND MCPHEE, N. F. *A Field Guide to Genetic Programming*. Lulu, 2008. With contributions by J. R. Koza. Available for free at http://www.gp-field-guide.org.uk.

[75] ROPKE, S., AND PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science 40*, 4 (2006), 455–472.

[76] ROSS, P. Hyper-heuristics. In *Search Methodolgies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Kluwer, 2005, pp. 529–556.

[77] ROTHLAUF, F., AND OETZEL, M. On the locality of grammatical evolution. In *Genetic Programming*, P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekàrt, Eds., vol. 3905 of *Lecture Notes in Computer Science*. 2006, pp. 320–330.

[78] RYAN, C., COLLINS, J., AND O'NEILL, M. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming* (1998), vol. 1391 of *Lecture Notes in Computer Science*, pp. 83–96.

[79] RYAN, D., HJORRING, C., AND GLOVER, F. Extensions of the petal method for vehicle routing. *Journal of the Operational Research Society 44*, 3 (1993), 289–296.

[80] SABAR, N., AYOB, M., KENDALL, G., AND QU, R. Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation 17*, 6 (2013), 840–861.

[81] SHAW, P. Using constraint programming and local search methods to solve vehicle routing problems. In *CP-98 Fourth international conference on principles and practise of constraint programming. Lecture notes in computer science* (1998), vol. 1520, pp. 417–431.

[82] SOLOMON, M. Algorithms for the vehicle routing problem with time windows. *Transportation Science 29*, 2 (1995), 156–166.

[83] SOUBEIGA, E. *Development and Application of Hyperheuristics to Personnel Scheduling*. PhD thesis, University of Nottingham, 2003.

[84] TAILLARD, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research 64*, 2 (January 1993), 278–285.

[85] THOMPSON, P., AND PSARAFTIS, H. Cyclic transfer algorithms for the multivehicle routing and scheduling problems. *Operations Research 41*, 5 (1993), 935–946.

[86] THORHAUER, A., AND ROTHLAUF, F. Structural difficulty in grammatical evolution versus genetic programming. In *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference* (2013), pp. 997–1004.

[87] TOTH, P., AND VIGO, D. *The Vehicle Routing Problem*. SIAM, 2002.

[88] WALKER, J., OCHOA, G., GENDREAU, M., AND BURKE, E. Vehicle routing and adaptive iterated local search within the HyFlex hyper-heuristic framework. In *Learning and Intelligent Optimization* (2012), vol. 7219 of *Lecture Notes in Computer Science*, Springer, pp. 265–276.

[89] WHIGHAM, P. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications* (1995), pp. 33–41.

[90] YAO, X., AND XU, Y. Recent advances in evolutionary computation. *Journal of Computer Science and Technology 21*, 1 (2006), 1–18.