Crossover Control in Selection Hyper-heuristics: Case Studies using MKP and HyFlex

John H. Drake, BSc (Hons), MSc

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

July 2014

Abstract

Hyper-heuristics are a class of high-level search methodologies which operate over a search space of heuristics rather than a search space of solutions. Hyper-heuristic research has set out to develop methods which are more general than traditional search and optimisation techniques. In recent years, focus has shifted considerably towards cross-domain heuristic search. The intention is to develop methods which are able to deliver an acceptable level of performance over a variety of different problem domains, given a set of low-level heuristics to work with.

This thesis presents a body of work investigating the use of selection hyper-heuristics in a number of different problem domains. Specifically the use of crossover operators, prevalent in many evolutionary algorithms, is explored within the context of single-point search hyper-heuristics. A number of traditional selection hyper-heuristics are applied to instances of a well-known NP-hard combinatorial optimisation problem, the multidimensional knapsack problem. This domain is chosen as a benchmark for the variety of existing problem instances and solution methods available. The results suggest that selection hyper-heuristics are a viable method to solve some instances of this problem domain. Following this, a framework is defined to describe the conceptual level at which crossover low-level heuristics are managed in single-point selection hyper-heuristics. HyFlex is an existing software framework which supports the design of heuristic search methods over multiple problem domains, i.e. cross-domain optimisation. A traditional heuristic selection mechanism is modified in order to improve results in the context of cross-domain optimisation. Finally the effect of crossover use in cross-domain optimisation is explored.

Acknowledgements

My sincerest thanks go to my supervisor Dr. Ender Özcan without whom this thesis would not be possible. The advice and support he has given me over the last few years has enabled me to get to where I am today. I am also grateful to my external supervisor Professor Edmund Burke and the LANCS initiative for supporting me throughout the course of my PhD.

A special mention is due to the academic and administrative members of the Automated Scheduling, optimisAtion and Planning (ASAP) research group, past and present. I have been lucky enough to have a good working and social relationship with many members of the group, for which my life as a PhD student has been a richer experience.

Finally, I would like to thank my family and friends for their unconditional support throughout my academic career and life in general. There is no way I would have reached this point if it wasn't for you.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background and Motivation

Optimisation problems often explore a search space which is too large to enumerate and exhaustively search for an optimal solution. Various heuristics and metaheuristics have been applied successfully to problems of this nature. One drawback of such approaches is the necessity to manually adapt the method used to solve different problem domains or classes of problem. Hyper-heuristics are a class of high-level search techniques which aim to raise the level of generality at which search methods operate [190]. Hyper-heuristics are broadly split into two main categories, those which select a low-level heuristic to apply from a set of existing heuristics and those which create new heuristics from a set of low-level components [28]. This thesis is concerned with the first category, those methodologies which select a low-level heuristic to apply at a given point in a search.

The objective of cross-domain heuristic search is to develop methods which are able to consistently find good quality solutions in multiple problem domains, using a given set of low-level heuristics. The HyFlex framework [28, 166], introduced chiefly to support the first Cross-domain Heuristic Search Challenge (CHeSC2011) [31], is used as a benchmark framework for many of the methods investigated in this thesis. HyFlex provides a common software interface to test the performance of high-level search strategies over multiple problem domains, and an increasing body of associated research with which to compare. Currently HyFlex supports six problem domains and provides a search space of low-level heuristics from four categories.

The multidimensional knapsack problem (MKP) is a well-studied combinatorial optimisation problem with roots in capital budgeting and project selection. The objective of the MKP is to maximise the profit obtained when selecting a subset of knapsack items, given a set of constraints on the knapsack capacities. Each item has an associated profit value and consumes a certain amount of resources in a number of dimensions, with a capacity set for total resource consumption in each dimension. A large number of researchers have used this problem domain as a benchmark to compare algorithm performance. In recent years, the MKP has become somewhat of a favoured testing ground for research which combines exact and (meta-)heuristic methods. Here the MKP is used as a case study to compare a number of selection hyper-heuristics.

There has been an increase in the number of theoretical studies investigating the effect of mixing strategies in evolutionary algorithms of late. Using 'asymptotic hitting time' as a performance measure, He et al. [98] showed that for simple (1+1) EAs, there is no benefit to using a single mutation operator over a 'mixed strategy' of multiple mutation operators. Moreover if the mutation operators within a mixed strategy EA are mutually complementary, there exists a mixed strategy EA which strictly outperforms any EA using a single mutation operator. Lehre and Özcan [134] presented an initial study analysing the expected run-time of selection hyper-heuristics. This study showed that mixing low-level heuristics is more efficient than using an individual low-level heuristic in some problem domains provided the right mixing distribution. Although this is restricted to selection hyper-heuristics based on a (1+1) EA solving simple problems, it is shown that mixing heuristics could lead to exponentially faster search than using individual heuristics on certain problem domains. He et al. [99] provided a formal definition of complementary metaheuristics, and the conditions under which mixed strategy EAs outperform 'pure strategy' EAs using a single search method. This work also compared population-based mixed strategy and pure strategy EAs solving the 0-1 knapsack problem. It is observed that using a mixed strategy EA could result in better solutions than a pure strategy EA in over three quarters of the problem instances tested. A key theme emerging from these studies is that mixing low-level heuristics is provably beneficial to a search process in at least some circumstances. This provides a reasonable justification for investigating methods which aim to utilise the strengths of different operators at different points of a search such as selection hyper-heuristics.

The No Free Lunch (NFL) theorem [214] shows that over the set of all possible problems, any two search algorithms will exhibit the same performance on average. Hyper-heuristic research aims to combine search algorithms, utilising the strengths of different search methods at different points of the search process. In practice this can lead to a hyper-heuristic being outperformed by a bespoke search method for a given problem, due to the initial overhead involved in 'learning' how to solve the problem at hand. Although this will mean some loss in solution quality obtained compared to some state-of-the-art method, this is offset by the gain in generality hyper-heuristics offer. Which of these aspects is more important will depend on the problem domain in question.

Crossover is a core operator in many evolutionary algorithms, inspired by its biological namesake, included in many hyper-heuristic frameworks such as HyFlex [166] and Hyperion [200]. The behaviour of crossover operators is a well-studied area in the field of evolutionary computation. In Genetic Algorithms, the canonical form of crossover combines two suitably fit solutions to yield a new solution which inherits genetic material from both. The building block hypothesis [102, 90] states that a Genetic Algorithm works well when short, relatively fit sub-strings (schemas or building blocks) are recombined to produce a higher-order solution of even greater fitness. It follows from the NFL theorem that there are different classes of functions where both crossover and mutation can outperform each other. Mitchell et al. [157] introduced a class of problems known as 'Royal Road' functions in order to try and analyse the type of landscape crossover operators perform well in. These simple functions were designed in a way that tried to exploit the nature of crossover, by isolating the highly-fit blocks (or schemas) which were needed for an optimal solution, however Forrest and Mitchell [72] showed that a random mutation hill climber could actually outperform a Genetic Algorithm on these functions. Jansen and Wegener [113] showed that it is possible to have a function which can be expected to be optimised in polynomial time using a Genetic Algorithm with crossover, whereas using evolution strategies based on only selection and mutation need expected exponential time. This work indicated that crossover can be beneficial in some cases and should not be completely dismissed when designing optimisation methods, however it did not explicitly make use of building blocks. Watson and Jansen [210] introduced a function that was not only solvable by a Genetic Algorithm in polynomial time on average and exponential time for a mutation-based algorithm, but also used building blocks. Doerr et al. [58, 59] provided the first theoretical proof of crossover being beneficial in a practical optimisation problem. Their studies showed that introducing a problem-specific crossover operator into a mutation-based evolutionary algorithm solving the all-pairs shortest path problem reduces the expected optimisation time.

Despite the inclusion of crossover in modern selection hyper-heuristic frameworks and the proven benefit of using such operators in certain problem domains, there has been little research effort into strategies managing crossover in selection hyper-heuristics. As mixing low-level heuristics and utilising crossover is provably beneficial in some problems, it is a natural research direction to investigate the use of crossover operators in selection hyper-heuristics.

1.2 Aims and Scope

The purpose of this thesis is to provide a contribution to hyper-heuristic research by investigating the use of crossover in selection hyper-heuristics and applying selection hyper-heuristics to a new problem domain. Firstly, generic hyper-heuristics are applied to the multidimensional knapsack problem in order to assess the suitability of hyper-heuristics as a solution method in this problem domain. Secondly, these methods are then hybridised with specific exact methods and used as a test-bed for crossover management at different conceptual levels. Thirdly a modified version of a well-known hyper-heuristic heuristic selection method is presented, with the intention of improving cross-domain performance. This mechanism is then used within a base hyper-heuristic to compare high-level crossover control mechanisms. Finally, a number of high-level crossover control mechanisms are compared within the hyper-heuristics developed in earlier chapters and a state-of-the-art selection hyper-heuristic.

1.3 Contributions

1.3.1 Academic publications produced

A number of academic publications have been produced as a result of completing the research presented in this thesis. These publications are listed in order of the relevant chapter in which this research is contained:

Chapter 5:

- John H. Drake, Ender Özcan and Edmund K. Burke. A Case Study of Controlling Crossover in a Selection Hyper-heuristic Framework with MKP. *Submitted to an international journal* 2013.

Chapter 6:

- John H. Drake, Ender Özcan and Edmund K. Burke. An Improved Choice Function Heuristic Selection for Cross Domain Heuristic Search. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Proceedings of Parallel Problem Solving From Nature (PPSN 2012), Part II*, Volume 7492 of *LNCS*, pages 307-316, Taormina, Italy, 2012. Springer.

In addition to the publications produced as a direct result of the work in this thesis a number of related papers have been published during this course of research. These are listed in chronological order of publication:

- John H. Drake, Matthew Hyde, Khaled Ibrahim and Ender Özcan. A Genetic Programming Hyper-Heuristic for the Multidimensional Knapsack Problem. *Proceedings of the 11th IEEE International Conference on Cybernetic Intelligent Systems (CIS 2012)*, pages 76-80, Limerick, Ireland, 2012. IEEE Press.

- Jerry Swan, John H. Drake, Ender Özcan, James Goulding and John Woodward. *Computer and Information Sciences III: 27th International Symposium on Computer and Information Sciences*, chapter A Comparison of Acceptance Criteria for the Daily Car-Pooling Problem, pages 447-483. Springer, 2013.

- John H. Drake, N. Kililis and Ender Özcan. Generation of VNS Components with Grammatical Evolution for Vehicle Routing. In Krzysztof Krawiec, Alberto Moraglio, Ting Hu, A. Sima Etaner-Uyar and Bin Hu, editors, *Genetic Programming - 16th European Conference (EuroGP 2013)*, Volume 7831 of *LNCS*, pages 25-36, Vienna, Austria, 2013. Springer.

- Warren G. Jackson, Ender Özcan and John H. Drake. Late Acceptance-based Selection Hyper-heuristics for Cross-domain Heuristic Search. *Proceedings of the 13th Annual Workshop on Computational Intelligence (UKCI 2013)*, pages 228-235, Surrey, UK, 2013. IEEE Press.

1.4 Thesis Structure

The structure of the rest of the thesis, in the form of a brief overview of each of the individual chapters, will be outlined in the following sections.

1.4.1 Chapter 2: Literature Review

Chapter 2 presents a detailed literature survey of hyper-heuristics with a focus on recent developments in selection hyper-heuristics. A clear classification of hyper-heuristic methods is given, providing a grounding for the methods used in the remainder of the thesis. An introduction to the HyFlex framework and crossover control in selection hyper-heuristics is provided.

1.4.2 Chapter 3: The Multidimensional Knapsack Problem

The multidimensional knapsack problem is a well-studied combinatorial optimisation problem formally introduced in Chapter 3. A number of exact and metaheuristic methods used previously to solve this problem and popular benchmark sets are discussed. This problem domain is used as a benchmark case study in many of the empirical investigations in this thesis.

1.4.3 Chapter 4: A Study of Selection Hyper-heuristics Applied to the Multidimensional Knapsack Problem

Chapter 4 applies a number of standard selection hyper-heuristics operating over a generic set of low-level heuristics to the MKP. This chapter serves as a preliminary study of the suitability of hyper-heuristics as a method to solve this problem, analysing the performance of particular *heuristic selection method - move acceptance criterion* pairings in selection hyper-heuristics.

1.4.4 Chapter 5: A Case Study of Controlling Crossover in a Selection Hyper-heuristic Framework with MKP

Chapter 5 details an investigation of crossover use in selection hyper-heuristics solving the MKP. The responsibility of managing the second input solutions for crossover is tested at two levels. Firstly, it is tested at the hyper-heuristic level with the high-level search strategy providing a second solution each time a crossover low-level heuristic is selected. Secondly it is tested at the domain level, where the solutions are managed at the same level as the low-level heuristics.

1.4.5 Chapter 6: An Improved *Choice Function* Heuristic Selection for Cross-Domain Heuristic Search

A modified version of a traditional single-point selection hyper-heuristic heuristic selection method is presented in Chapter 5. The modified version of the *Choice Function* is designed to overcome some of the shortfalls of the standard *Choice Function* when used for cross-domain optimisation.

1.4.6 Chapter 7: Crossover Control in Cross-domain Optimisation

Chapter 7 investigates the introduction of the crossover management techniques of Chapter 5 into the selection hyper-heuristic proposed in Chapter 6. The crossover management scheme of a state-of-the-art selection hyper-heuristic is also investigated, firstly by introducing it into an existing selection hyper-heuristic and secondly by replacing it within the hyper-heuristic from which it is taken.

1.4.7 Chapter 8: Conclusion

The final section provides a summary of the work undertaken in the thesis and the contributions contained therein. An outline of potential future research directions and discussion for extensions to the existing work is given.

CHAPTER 2

Hyper-heuristics

2.1 Introduction

This chapter will overview the related hyper-heuristics literature for the work contained within this thesis, providing some context to the contributions made. The literature review is split into four main areas. These areas are: metaheuristics, hyper-heuristics, recent developments in selection hyper-heuristics and the HyFlex framework.

Section 2.2 gives a definition of metaheuristics and a brief description of some well-studied metaheuristic techniques. An introduction to, and classification of, hyper-heuristic approaches is then presented in Section 2.3. The term hyper-heuristic is used to define a method which operates on a search space of low-level heuristics or heuristic components rather than a search space of solutions. A formal definition and brief history of hyper-heuristics is given. The two main categories of hyper-heuristics, selection hyper-heuristics and generation hyper-heuristics, are introduced. The hyper-heuristics used in this thesis are contained within the former category. Following this a detailed description of recent developments in selection hyper-heuristics is given in Section 2.4. Categorisation of the recent selection hyper-heuristics discussed are provided by heuristic selection method, move acceptance criterion and problem domain used in Appendix A.

Hyper-heuristic research aims to 'raise the level of generality' at which search methods operate. One flavour of generality is the application of search methods across multiple problem domains. Recently, considerable research effort has gone into developing selection hyper-heuristics for cross-domain optimisation. An increased amount of attention is now given to methods which ayre able to perform well over a variety of problems. The HyFlex [27, 166] framework was developed to support the first Cross-domain Heuristic Search Challenge (CHeSC2011) [31] and standardise research into heuristic search methods operating over multiple problem domains. Section 2.5 introduces this framework and describes methods from the literature developed using HyFlex. The problem domains contained within HyFlex are briefly introduced with references for further reading provided where necessary.

The main focus of this thesis is to investigate the management of low-level heuristics such as crossover, which require more than one solution as input, within single-point search hyper-heuristic frameworks. Hyper-heuristic research provides a distinct separation between the heuristic search space and the solution search space, isolating the high-level heuristic search method from the problem domain it is being applied to. A conceptual framework is proposed, defining the process of managing the solutions for input to low-level heuristics requiring multiple solutions at either the hyper-heuristic level or the problem domain level. Experimental results suggest that, in at least some problem domains, it is preferable to manage input solutions at the problem domain level. Due to the nature of the separation between heuristic search space and solution search space, it is not always possible to manage input solutions at the problem domain level. As a result, different strategies for managing the input solutions at the hyper-heuristic level are tested, with results suggesting an instance-specific relationship between the strategy used to manage input solutions and performance in terms of objective value. In this chapter a context for the work presented in the remainder of the thesis is provided.

2.2 Metaheuristics

Many real-world optimisation problems create a search space which is so large that it becomes impractical to exhaustively search all possible states. Heuristics, in the field of optimisation, seek to find good quality solutions to a problem given a 'rule of thumb', using some intuition with no guarantee of solution quality. In essence, a *metaheuristic* is a high-level strategy which is designed to guide a computational search for any problem where heuristics exist to modify the current state of a given solution. Traditional heuristics are liable to become trapped in local, suboptimal areas of the search space, never to discover the globally optimal solution. Metaheuristics are designed to escape local optima, often offering a good trade-off between computational effort and solution quality. The term metaheuristic was first used by Glover [86] to describe Tabu Search and has recently been defined by Sörensen and Glover [199] as:

'...a high-level problem-independent algorithmic framework that provides

> a set of guidelines or strategies to develop heuristic optimization algorithms. ... A problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in a metaheuristic framework is also referred to as a metaheuristic.'

This definition includes many popular techniques such as evolutionary algorithms, Simulated Annealing, Tabu Search and many more. Some of the most popular techniques, including some which will be used in the remainder of this thesis are described in the following sections.

2.2.1 Local Search

Local search methods are a relatively simple class of metaheuristics, based on the concept of locality between candidate solutions for a given problem. A typical local search algorithm moves through the search space from one solution to another within a neighbourhood, where a neighbour is defined as any state that can be reached from the current solution through some modification. In the case that a local search method will only move from one solution to another when that move results in some improvement with respect to a particular objective function, it is referred to as hill climbing.

2.2.2 Evolutionary Algorithms

Evolutionary algorithms are a class of search techniques inspired by the natural process of evolution, of which by far the most well-known are Genetic Algorithms [90]. A Genetic Algorithm iteratively updates a population of solutions through the use of operators which modify a solution (mutation) or recombine multiple solutions (crossover). A simple Genetic Algorithm is outlined in Algorithm 1.

Algorithm 1 Pseudocode of a simple Genetic Algorithm

```
1: generate initial population of solutions
2: repeat
3:     calculate fitness of each solution in the population
4:     select suitably fit solutions to breed
5:     apply crossover with a given probability
6:     apply mutation with a given probability
7: until termination criterion is met
8: return best solution
```

Memetic Algorithms [159, 160] are a hybrid approach to problem solving which combine evolutionary algorithms (often Genetic Algorithms) with local search techniques in order to improve the fitness of each individual in the population. The term Memetic Algorithm was first coined by Moscato [159] to cover a number of techniques using the idea of a *meme* [55] as a 'unit of cultural transmission'. A simple Memetic Algorithm introduces a local search phase into a Genetic Algorithm after crossover and mutation have (potentially) been performed during the evolutionary process.

2.2.3 Tabu Search

Tabu Search [87, 89] is another metaheuristic technique designed to overcome the problem of escaping local optima in local search. Tabu Search makes use of a memory to guide the algorithm to areas of the search space it has not previously visited. The memory makes a note of recently visited areas of the search space and places them on a list of tabu (taboo) moves, i.e. the algorithm is not allowed to visit these points when moving to the next solution. This hopefully leads the search away from local optima and towards solutions closer to the global optimum. Algorithm 2 shows the basic outline of Tabu Search.

Algorithm 2 Pseudocode of Tabu Search

```
1: select an initial candidate solution $S$
2: repeat
3:     find the best neighbour solution $S'$ adjacent to $S$ not in tabu list
4:     add $S'$ to tabu list
5:     if $S'$ is a 'better' solution than $S$ then
6:         $S \leftarrow S'$
7:     end if
8: until termination criterion is met
9: return $S$
```

Tabu Search only tends to make negative moves when stuck in local optima, unlike Simulated Annealing which can make negative moves at any time. Tabu Search has been successfully implemented as a metaheuristic for a large number of application areas including examination timetabling [80], vehicle routing [83], the travelling salesman problem [87] and job shop scheduling [87].

2.2.4 Simulated Annealing

Simulated Annealing [127] is a stochastic metaheuristic technique for combinatorial optimisation problems. Analogous to the process of annealing in metallurgy, Simulated Annealing probabilistically decides whether to accept a change to a given system based on a temperature parameter. The temperature parameter is initially set to a high value which is gradually reduced as the system 'cools' throughout the search process. In general, Simulated Annealing will accept non-improving moves with greater probability at the beginning of a search whilst the temperature is high. As the temperature reduces the system will accept non-improving moves at a much lower rate. In the case where a degradation in solution quality occurs, a move is accepted probabilistically based on the current value of the temperature parameter, and the level of change in solution quality. A number of measures have been proposed in the literature to calculate this probability. Where Simulated Annealing is used in this thesis, the probability p is defined as:

$$p = \frac{1}{1 + e^{-\Delta/T}} \tag{2.2.1}$$

where Δ is the change in objective function value and T is the current temperature value. The acceptable level of worsening solutions will be reduced as the temperature decreases and the probability of moving to a lower quality solution will lessen over time. The basic outline of Simulated Annealing is shown in Algorithm 3.

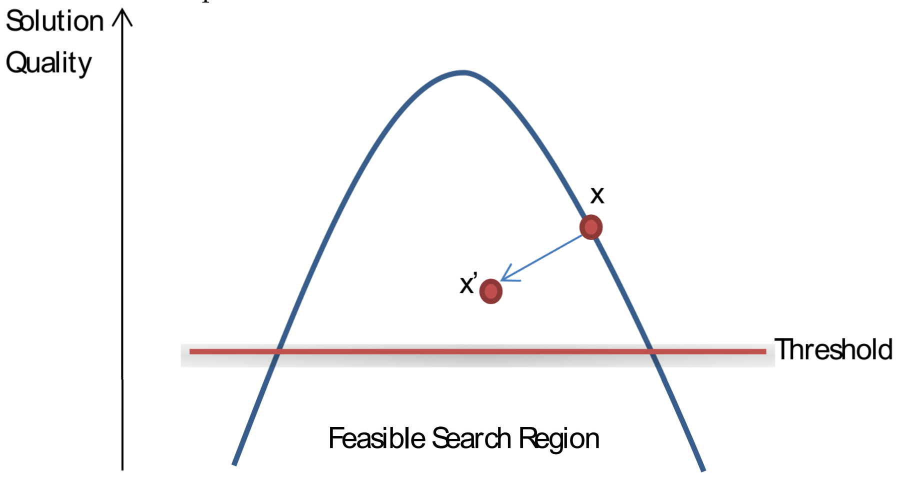
2.2.5 Great Deluge

Great Deluge is a general purpose hill climbing optimisation technique introduced by Dueck [65], inspired by the notion of a person on a hill escaping flood water (i.e. a 'Great Deluge'). The analogy is that in a great deluge, a person will try to move in any direction which avoids a rising water level in the hope of finding higher ground. As shown in Figure 2.1, a move between x and x′ is permitted provided it is above (in the case of a maximisation problem) a certain threshold which is gradually raised during the search process.

Algorithm 3 Pseudocode of Simulated Annealing

```
1: select an initial candidate solution $S$, initialise $T$
2: repeat
3:     find a neighbour solution $S'$ adjacent to $S$
4:     $\Delta \leftarrow$ difference in quality between $S'$ and $S$
5:     if $S'$ is a 'better' solution than $S$ then
6:         $S \leftarrow S'$
7:     else
8:         $r \leftarrow$ random number between [0,1]
9:         if $p < r$ then
10:            $S \leftarrow S'$
11:        end if
12:        decrease $T$
13:    end if
14: until termination criterion is met
15: return $S$
```

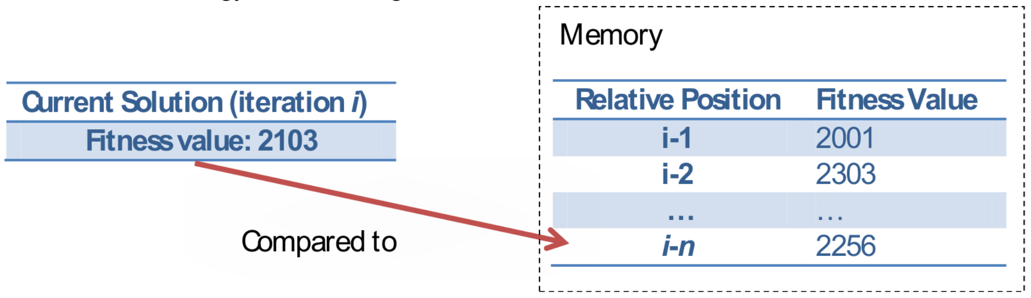
Figure 2.1: An abstract depiction of Great Deluge hill climbing in the case of a maximisation problem



2.2.6 Late Acceptance Strategy

Late Acceptance Strategy is a hill climbing metaheuristic strategy proposed by Burke and Bykov [20]. Late Acceptance Strategy promotes a general trend of improvement throughout the search process, by comparing a candidate solution to a solution generated a specified number of steps before kept in memory. In its most basic form Late Acceptance Strategy is an extension of simple greedy hill climbing. In hill climbing, a solution is accepted if it is of better quality than the one immediately preceding it. Late Acceptance Strategy accepts a solution if it is of better quality than the previous solution or the solution n iterations previously, where n is the size of a memory of previously seen solutions. An accepted solution replaces the solution it is compared with in the memory of previous solutions. In the case of a non-accepted solution, the solution of n iterations previously is replaced by the solution of the previous iteration (i.e. last accepted solution). The entire list of n previously seen solutions is initialised to the value of the starting solution at the beginning of the optimisation process. In the example given in Figure 2.2, assuming a minimisation problem, the new solution will be accepted as it has a fitness value strictly less than the solution kept in memory from n iterations past. In the case of a maximisation problem, this solution will also be accepted as it has a fitness value greater than its immediate predecessor in the memory ($i-1$). In both cases it will replace the n-th previous solution ($i-n$) in the memory.

Figure 2.2: Comparison of current solution to solution in memory in Late Acceptance Strategy hill climbing



2.3 A survey of hyper-heuristics

This section introduces the field of hyper-heuristics and is split into four sections. Section 2.3.1 briefly describes the history of hyper-heuristics and introduces the standard definitions used in hyper-heuristic research. Section 2.3.2 provides a classification of hyper-heuristic approaches, distinguishing between the main categories of hyper-heuristic methodologies. Section 2.3.3 and Section 2.3.4 provide a more detailed introduction to selection hyper-heuristics and generation hyper-heuristics respectively.

2.3.1 Hyper-heuristic history and definitions

Hyper-heuristics are high-level search methodologies which aim to solve computationally difficult problems. Unlike traditional techniques, a hyper-heuristic operates on a search space of heuristics rather than directly on the search space of solutions. The term 'hyper-heuristic' was first used by Denzinger et al. [57] to describe a technique which selects and combines a number of artificial intelligence methods. Although the term hyper-heuristic was first used at this time, ideas which exhibited hyper-heuristic behaviour can be traced back as early as 1961 in the field of job shop scheduling [70], where combining scheduling rules was shown to outperform taking any of the rules individually. The term hyper-heuristic was first used in the field of combinatorial optimisation by Cowling et al. [47], and was defined as '*heuristics to choose heuristics*'. This paper investigated the application of a number of *Simple Random*, *Greedy* and *Choice Function*-based hyper-heuristic approaches to a real-world sales summit scheduling problem using two deterministic move acceptance criteria: *All Moves* and *Only Improving*. The *Choice Function* heuristic selection method will be introduced in detail in Section 2.3.3. In the first journal article to appear using the term, Burke et al. [22] presented a *Tabu Search*-based hyper-heuristic. In this system, a set of low-level heuristics are ranked using rules based on Reinforcement Learning and compete against each other for selection. The hyper-heuristic selects the highest ranked heuristic not present in the tabu list. If an improvement is made after applying the selected heuristic its rank is increased, if not, its rank is decreased and it is placed in the tabu list until the current solution has changed. This hyper-heuristic was applied to nurse scheduling and university course timetabling problems obtaining competitive results. Hyper-heuristics have since been applied successfully to a wide range of problems such as examination timetabling [22, 175, 176, 35, 192], production scheduling [70], nurse scheduling [22, 166], bin packing [166, 135], sports scheduling [84], dynamic environments [124, 126] and vehicle routing [166, 79].

Research trends have lead to a number of different hyper-heuristic approaches being developed, particularly those concerned with automatically generating new heuristics, for which the original definition of a hyper-heuristic is too limited to cover. A more general definition is offered by Burke et al. [29, 36]:

> 'A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems.'

This more general terminology includes systems which use high-level strategies other than heuristics within the definition of hyper-heuristics and covers the two main classes of hyper-heuristics, those concerned with heuristic selection and those with heuristic generation. Burke et al. [36] also identified a number of closely related areas to hyper-heuristic research including: Adaptive Operator Selection [68, 148], Reactive Search [10, 11], Variable Neighbourhood Search [163], Adaptive Memetic Algorithms [169] and Algorithm Portfolios [106]. Although an overview of these methods is not directly provided in this thesis, a number of the approaches discussed will overlap with some of these areas. The references provided are a good starting point for each of these techniques for the interested reader.

Although often considered as an alternative to metaheuristics, the recent definition of a metaheuristic offered by Sörensen and Glover [199] somewhat subsumes hyper-heuristics. According to this definition, a selection hyper-heuristic is a metaheuristic which provides a framework within which to mix and control low-level heuristics, whereas a generation hyper-heuristic is a metaheuristic to generate heuristics. It follows that if a metaheuristic is a heuristic, a hyper-heuristic can either be a metaheuristic itself (e.g. a Genetic Programming system to generate heuristics [107]) or contain metaheuristic components (e.g. a selection hyper-heuristic using *Simulated Annealing* move acceptance criterion [14]).

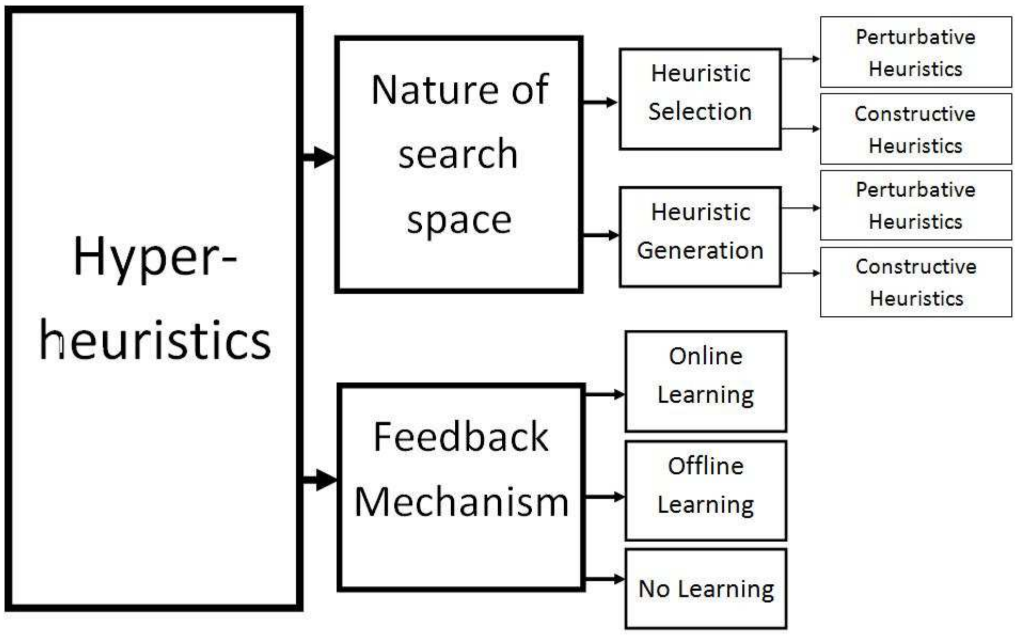
2.3.2 A classification of hyper-heuristic approaches

There are two main categories of hyper-heuristics outlined by Burke et al. [29]. The first category contains those methodologies which select which low-level heuristic to apply at a given position in the search space from a set of existing heuristics. The second contains methodologies which create new heuristics from a set of components of other existing low-level heuristics [28]. These categories are then further broken down to distinguish between hyper-heuristics which construct solutions (constructive) [25] and those which aim to improve complete solutions through local search (perturbative) [175].

Aside from the nature of the search space, many hyper-heuristics learn from feedback given regarding heuristic performance to guide low-level heuristic choice or generation. Such feedback is used to learn in one of two ways: online learning occurs during the process of solving a problem instance and adapts continuously throughout the process [173]; in offline learning, a system is trained on a subset of problems prior to full execution in order to assert a set of rules to apply to unseen instances [107]. This taxonomy of learning behaviours is congruent with the general classification of parameter setting in evolutionary algorithms provided by Eiben et al. [66]. The definition of online and offline learning is parallel to the differentiation between parameter control and parameter tuning given in this paper.

The classification of hyper-heuristic components as defined by Burke et al. [29] is summarised in Figure 2.3. A given hyper-heuristic will generally belong to one of the four subcategories of heuristic selection and heuristic generation and use one of the three feedback mechanisms outlined. Section 2.3.3 and Section 2.3.4 provide an introduction to selection and generation hyper-heuristics respectively.

Figure 2.3: A high-level overview of hyper-heuristic components



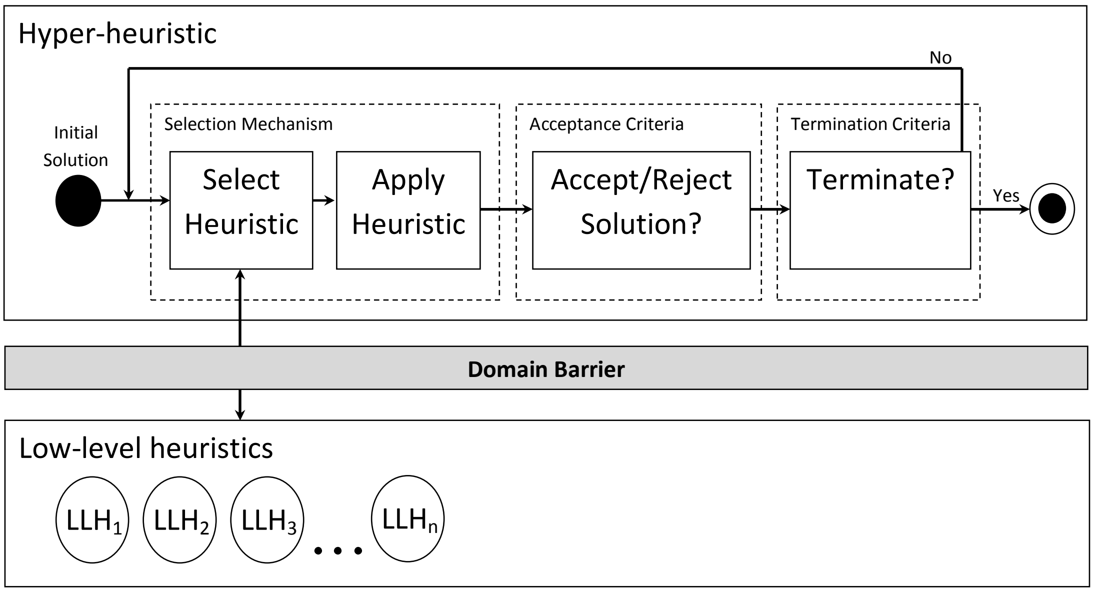
2.3.3 Heuristics to select heuristics

Selection hyper-heuristics operate over a set of low-level heuristics, selecting a heuristic to apply at each point of a search in a domain-agnostic way. This section introduces a number of traditional selection hyper-heuristic frameworks and some of the popular mechanisms and components used by methods operating within these frameworks.

Depending on the number of solutions used during the search process, a distinction is made between approaches performing *single-point-based* search using a single current solution and *population-based* (multi-point-based) search using multiple candidate solutions. The traditional single-point-based search hyper-heuristic framework relies on two key components, a heuristic selection method and a move acceptance criterion as decomposed by Özcan et al. [173]. Operating on a single solution, low-level heuristics are repeatedly selected and applied, with a decision made as to whether to accept a move at each step until some termination criterion is met. Such hyper-heuristics are labelled *heuristic selection method - move acceptance criterion* hereafter in this thesis. This general framework is illustrated in Figure 2.4.

The heuristic selection method chooses which heuristic to apply at a given time from the set of n available low-level heuristics $LLH_1, ..., LLH_n$. A new candidate solution is then obtained by applying the selected heuristic. A move acceptance criterion is used

Figure 2.4: Classic single-point search hyper-heuristic framework



to decide whether to accept or reject the new solution. If accepted, the new candidate replaces the original solution. Most of the hyper-heuristics developed in this thesis are of this nature. This approach sits firmly in the category of selection hyper-heuristics operating on perturbative heuristics. The learning techniques used will vary depending on the heuristic selection mechanism used. Table 2.1 and Table 2.2 provide definitions for some of the common heuristic selection methods used in the remainder of the thesis, while Table 2.3 provides definitions for a number of common move acceptance criteria.

In their initial hyper-heuristic work Cowling et al. [47] experimented with a number of heuristic selection mechanisms, including *Simple Random* and *Choice Function*, using two simple move acceptance criteria, accepting *All Moves* and *Only Improving* moves. In this early work, the *Choice Function* selection combined with *All Moves* acceptance was shown to work well when applied to a sales summit scheduling problem.

Reinforcement Learning [115] is frequently used as a heuristic selection method in selection hyper-heuristics. Nareyek [162] analysed a number of weight adaptation functions and two simple heuristic selection methods within *Reinforcement Learning* for heuristic selection. Taking the heuristic with the maximum utility value, and using simple additive (+1) and subtractive (-1) weight adaptation, were shown to be reasonable choices when using *Reinforcement Learning* as a heuristic selection method in selection hyper-heuristics for constraint satisfaction problems (CSP).

Great Deluge [65] is a metaheuristic which has shown to be a promising acceptance criterion in hyper-heuristics. In hyper-heuristics, Bilgin et al. [14] found that a hyper-

Table 2.1: Key definitions of heuristic selection methods

Selection Method	Description
Choice Function (CF) [47]	Heuristics are chosen based on a combination of three different measures. The first measure (f_1) records the previous performance of each individual heuristic, with more recent executions carrying larger weight. The value of f_1 for each low-level heuristic $h_1, h_2, ..., h_j$ is calculated as: $$f_1(h_j) = \sum_n \alpha^{n-1} \frac{I_n(h_j)}{T_n(h_j)} \quad (2.3.1)$$ where $I_n(h_j)$ and $T_n(h_j)$ are the change in evaluation function and the time taken to call the heuristic respectively for each previous invocation n of heuristic h_j and α is a value between 0 and 1 giving greater importance to recent performance. The second measure (f_2) attempts to capture any pair-wise dependencies between heuristics. Values of f_2 are calculated for each heuristic h_j when invoked immediately following h_k using the formula in Equation 2.3.2: $$f_2(h_k, h_j) = \sum_n \beta^{n-1} \frac{I_n(h_k, h_j)}{T_n(h_k, h_j)} \quad (2.3.2)$$ where $I_n(h_k, h_j)$ and $T_n(h_k, h_j)$ are the change in evaluation function and the time taken to call the heuristic respectively for each previous invocation n of heuristic h_j following h_k and β is a value between 0 and 1 which also gives greater importance to recent performance. The third measure (f_3) is the time elapsed ($\tau(h_j)$) since a heuristic was last selected by the Choice Function. This gives all heuristics a positive chance of selection. $$f_3(h_j) = \tau(h_j) \quad (2.3.3)$$ In order to rank heuristics a score is given to each heuristic with Choice Function F calculated as: $$F(h_j) = \alpha f_1(h_j) + \beta f_2(h_k, h_j) + \delta f_3(h_j) \quad (2.3.4)$$ where α and β as defined previously weight f_1 and f_2 respectively to provide intensification of the heuristic search process whilst δ weights f_3 to provide sufficient diversification.

Table 2.2: Key definitions of heuristic selection methods (ii)

Selection Method	Description
Reinforcement Learning (RL) [115]	All low-level heuristics are given a utility weight. If a heuristic improves a solution this weight is increased by an amount defined by the chosen adaptation function, conversely if a heuristic does not improve a solution this weight is decreased accordingly. Heuristic selection at the next step of the search is then based on these values. This selection can be done in many ways, for example it is possible to choose randomly between the heuristics with the largest utility weight, or to select a heuristic with a probability proportional to its utility weight
Simple Random (SR) [47]	Selects a heuristic randomly from the set of available low-level heuristics at each point in the search

Table 2.3: Key definitions of move acceptance criteria

Move Acceptance Criteria	Description
All Moves (AM) [47]	Accepts any move made following the application of a heuristic, regardless of change in solution quality
Only Improving (OI) [47]	Accepts any move made following the application of a heuristic which yields an improvement in solution quality
Great Deluge (GD) [115]	When used as an acceptance criterion, Great Deluge always accepts improving moves and accepts worsening moves which are below (in the case of a minimisation problem) a certain threshold which is reduced over time at a linear rate (see Section 2.2.5 for more information)
Simulated Annealing (SA) [127]	Any move resulting in a solution of equal or greater quality than the previous move is accepted. If a move yields a solution of poorer quality, the move is accepted probabilistically based on the level of degradation in solution quality and the current temperature (a parameter which decreases over time). Please refer to Section 2.2.4 for more information
Late Acceptance Strategy (LAS) [20]	Accepts a move if it is an improvement on the solution observed a pre-determined number of steps previously held in memory. In the case of a non-improving move, the solution in memory is replaced by the last solution accepted. Further details can be found in Section 2.2.6

heuristic without learning (*Simple Random*) performed comparably to (and often better than) more 'advanced' heuristic selection mechanisms (*Choice Function* and *Tabu Search*) when *Great Deluge* was used as a move acceptance criterion over a wide range of timetabling benchmarks. Özcan et al. [176] extended this work and applied a *Reinforcement Learning - Great Deluge* hyper-heuristic to the same set of benchmark instances, again providing good results. Kendall and Mohamad [120] used a slight variation of the *Great Deluge* algorithm (record-to-record travel [65]) and also found it to be a promising acceptance criterion when solving a telecommunications network problem.

Simulated Annealing [127] is another generic metaheuristic technique for optimisation often used as an acceptance criterion in hyper-heuristics. *Choice Function - Simulated Annealing* was found to be the best of thirty-five different *heuristic selection method - move acceptance criterion* combinations over a range of examination timetabling benchmarks by Bilgin et al. [14].

Late Acceptance Strategy [20] is a relatively new general purpose optimisation strategy. *Late Acceptance Strategy* has been shown to perform as well as other powerful optimisation methods including *Simulated Annealing* and *Great Deluge* [175, 20], but only relies on the setting of a single parameter, the length of the memory. Although only one parameter is required, it is still important that this parameter is set appropriately. If it is too short the search will converge on a sub-optimal point quickly, if the memory is too long the search will stagnate. Özcan et al. [175] experimented using *Late Acceptance Strategy* as a move acceptance criterion within a single-point hyper-heuristic framework to solve standard benchmarks of the examination timetabling problem. This work suggested that *Late Acceptance Strategy* was relatively successful when used with *Simple Random* selection in the context of examination timetabling, it was unsuitable when used with more intelligent heuristic selection methods such as *Choice Function* and *Reinforcement Learning*.

The work contained in this thesis uses many single-point search hyper-heuristics such as those described here. A large number of other heuristic selection mechanisms and move acceptance criteria exist in the literature. As a complete description of all heuristic selection mechanisms and move acceptance criteria is beyond the scope of this chapter, a number of survey papers [190, 38, 36] provide a thorough grounding in this area.

Selection hyper-heuristic frameworks

Özcan et al. [174] describe and compare four different selection hyper-heuristic frameworks: F_A, F_B, F_C and F_D. F_A is the traditional selection hyper-heuristic framework where a low-level heuristic is selected and applied, then subsequently accepted or rejected based the quality of the move, as shown in Figure 2.4 previously. F_B selects a low-level heuristic from a set of mutational heuristics and hill climbers. If a mutational heuristic is selected, a hill climber is then also applied before a decision is made whether to accept or reject the move. F_C selects and applies a mutational heuristic followed by a pre-defined hill climber, before deciding whether or not to accept a move. F_D separates mutational heuristics and hill climbers into two distinct groups. A mutational heuristic from the first group is chosen and applied with the move accepted or rejected based on performance. A hill climber is then chosen from the second group and a separate decision is made whether to accept or reject the move. A number of these frameworks are used in this thesis and are discussed further in the relevant sections.

Controlling crossover in selection hyper-heuristics

Despite the inclusion of crossover operators in modern hyper-heuristic frameworks such as HyFlex [166] and Hyperion [200], limited research effort has been directed at managing the input for this type of low-level heuristic. Typically a selection hyper-heuristic operates within a single-point search framework. In the case of low-level heuristics which require more than one solution as input, a natural choice for one of the solutions is the current working solution in the hyper-heuristic. Unfortunately there is not necessarily a natural choice for which solution to use as the second input to such operators. In the CHeSC2011 competition [165] based on the HyFlex framework, only two of the top ten entrants provided a description of a strategy to control input solutions for crossover operators. Of the two hyper-heuristics which do provide a description, one simply uses the best seen solution so far as the second input and the other, the eventual competition winner [155], provides a detailed explanation of a crossover management scheme. This hyper-heuristic maintains a memory of the 5 best solutions seen so far, of which a random solution is used each time a crossover low-level heuristic is chosen. When a new best-of-run solution is found it replaces one of the 5 solutions in memory, again chosen at random. The HyFlex framework is introduced in detail in Section 2.5.

Kheiri and Özcan [122] used a simple scheme to manage the second inputs for low-level heuristics, again using the HyFlex framework. A circular queue containing the best solutions seen so far is maintained to provide second inputs for crossover operators, however the length of the queue is set arbitrarily. A pointer indicates which solution is to be used each time a crossover heuristic requires a second input solution, which is then advanced to the next solution in the queue following each application of crossover.

The methods discussed above relate to hyper-heuristics which manage the inputs for multi-input operators at the hyper-heuristic level. That is to say, the high-level search methodology is responsible for providing a second input when a low-level heuristic such as crossover is selected. Maturana et al. [149] selected a crossover operator to use at each step in evolutionary algorithms solving the satisfiability problem (3-SAT). Although the choice of crossover operator is made at the hyper-heuristic level, the selection of input solutions for the crossover operator chosen is performed at the domain level. The crossover operators available all require two individuals as input, with two schemes used to select these individuals. In the early experimentation, this selection is performed randomly between all individuals in the population. A fitness-biased selection scheme is also used however the details of this mechanism are not described in detail.

Cobos et al. [43] presented two selection hyper-heuristics operating over a set of meta-heuristics including Genetic Algorithm variants. Rather than the single-point search framework used by Kheiri and Özcan [122] and Misir et al. [155], the low-level heuristics in this framework work with a shared population of solutions. The Genetic Algorithm variants perform crossover on two individuals selected from this shared population. In this case, the responsibility for providing the two inputs necessary for crossover is below the domain barrier, managed by the low-level heuristics, rather than at the hyper-heuristic level.

Ren et al. [188] used Ant Colony Optimisation and a Genetic Algorithm to evolve sequences of low-level heuristics to be applied to instances of the p-median problem. Included in the set of low-level heuristics was the crossover operator of Correa et al. [45]. Although this work stated that this crossover operator 'requires two input solutions', each sequence of low-level heuristics is only applied to a single solution. How the second input solution required for crossover is managed is not defined.

2.3.4 Heuristics to generate heuristics

Aside from selection hyper-heuristics there is also considerable research interest in the field of automated heuristic generation. Techniques in this area seek to automatically create heuristics which are competitive with human-designed heuristics. Evolutionary algorithms are commonly used to search the space of low-level heuristic components in order to automatically design new heuristics. The first part of this section describes the use of Genetic Programming to evolve heuristics. Following this, recent work focussed on evolving policy matrices to solve combinatorial optimisation problems is introduced. Grammatical Evolution is another branch of evolutionary computation whereby solutions are evolved using a pre-defined grammar. A number of studies which use Grammatical Evolution to create low-level heuristics are then discussed. Finally, some other recent related techniques are discussed, which do not fit strictly under the definition of generation hyper-heuristics.

Genetic Programming as a hyper-heuristic to generate heuristics

Genetic Programming is one of the more recently developed class of evolutionary algorithms proposed by Koza [129]. Unlike traditional forms of evolutionary computation, populations of computer programs (usually expressed as tree structures although other representations are possible) are evolved using the naturally inspired notions of inheritance (crossover), selection and variation (mutation). Rather than producing fixed-length encoded representations of candidate solutions to a given problem, the evolved program itself (when executed) is the solution. Genetic Programming can be seen as an extension of Genetic Algorithms with only the representations separating the two classes, however there are enough differences to consider the two as separate techniques. A standard pseudocode for a typical Genetic Programming system is shown in Algorithm 4.

Algorithm 4 Pseudocode of Genetic Programming

```
1: generate initial population of programs
2: repeat
3:     calculate fitness of each program in the population
4:     select suitably fit programs to breed
5:     create new programs by performing genetic operations on selected individuals
6: until acceptable solution is found
7: return best solution
```

Burke et al. [28] outline the suitability of Genetic Programming as a hyper-heuristic to generate new heuristics and provide a survey of previous work creating heuristics using Genetic Programming. Some of the main advantages of using Genetic Programming as a hyper-heuristic in such a way highlighted in this paper are:

- Genetic Programming relies on expert knowledge to define its terminal and function sets (variables to define the state of the problem and operators to combine them). As human expert knowledge is necessary, domain-specific information can be incorporated into the fundamental components of the system.
- Whilst some other methods such as Genetic Algorithms may restrict the length

of an encoded solution in order to facilitate simple genetic operators, Genetic Programming trees have variable length representation. This can be useful if the best length encoding for heuristic representation is not known and the effects of code bloat are controlled.

- A functional programming language such as LISP is naturally suited to representing data in a tree structure. This property allows Genetic Programming to evolve executable programs (i.e. heuristics).

Genetic Programming has successfully been used to evolve new constructive heuristics comparable to human designed heuristics by Burke et al. for strip packing [30] and bin packing problems [24, 26, 34] and to create dispatching rules for the job shop scheduling problem by Geiger et al. [82]. Bader-El-Den and Poli [7] used Genetic Programming to quickly generate 'disposable' heuristics to solve the satisfiability problem. Again, this work generated heuristics comparable to human designed ones however only a limited search space of heuristics was covered. Fukunaga [74, 75, 76] also used Genetic Programming to generate local search heuristics for the satisfiability problem. Hauptman et al. [97] presented a Genetic Programming system to generate solvers for two common puzzles including the NP-Complete FreeCell puzzle. At a higher level of abstraction, Hyde et al. [109] used Genetic Programming to evolve the acceptance criterion component of a selection hyper-heuristic. The evolved acceptance criterion performed well when compared to standard acceptance criteria from the literature on instances of both bin packing and MAX-SAT. Drake et al. [61] employed a Genetic Programming system to evolve reusable constructive heuristics for the MKP. The evolved heuristics were able to yield human-competitive results, outperforming a number of traditional human designed constructive heuristics. Hong et al. [103] evolved mutation operators for an Evolutionary Programming system in the form of probability distributions using Genetic Programming. The evolved distributions were able to outperform two standard distributions: Gaussian and Cauchy, over a set of function optimisation problems.

Grammatical Evolution as a hyper-heuristic to generate heuristics

Grammatical Evolution (GE) [191] is another recently developed class of evolutionary algorithm closely related to Genetic Programming. Grammatical Evolution was introduced to address some of the issues present when using Genetic Programming. Any function and terminal set in Genetic Programming can be expressed in an equivalent Grammatical Evolution grammar. Grammatical Evolution aims to separate the genotype and phenotype of an individual in the population in a similar way to nature, something which is not done in Genetic Programming. In Genetic Programming, the search process is performed on the same component which the fitness function evaluates over (i.e. Genetic Programming trees). In Grammatical Evolution the search process is performed over a binary string (the genotype) and the fitness function evaluates the program (the phenotype) which is obtained. There are a number of advantages to approaching the search process in this way. Any strategy that operates on binary strings can be used to perform the search, this is not strictly limited to evolutionary approaches. In addition, the search is not limited by tree structure and the need to ensure solutions are valid, nor compromised by code 'bloat' in the same way it is in Genetic Programming. The genotype is processed as codons and mapped into phenotypes using a specified grammar. In biology, a codon describes a component of a protein. In Grammatical Evolution a specified number of bits representing a codon is converted to an integer, defining which rule will be chosen in a given part of the sequence. A grammar defining the structure of the desired output from the Grammatical Evolution is expressed in Backus Naur Form (BNF) in the form of production rules. BNF has the advantage of being language independent and only minor changes need to be made to a grammar to alter the search space. If the production rules specify a set of heuristics or heuristic components Grammatical Evolution can be used as a hyper-heuristic to generate heuristics.

Recently Burke et al. [33] have used Grammatical Evolution to generate low-level heuristics for bin packing. This paper generates heuristics which can consistently obtain solutions which use only one bin more than the optimal lower bound and often the optimal number of bins itself. Grammatical Evolution was also seen to be flexible enough to generate different move operators for different classes of bin packing problems as appropriate. Keller and Poli [118, 119] also use a grammar-based Genetic Programming system to evolve solvers for the travelling salesman problem. Drake et al. [63] used Grammatical Evolution to evolve both constructive and perturbative heuristics embedded in a Variable Neighbourhood Search framework to solve the vehicle routing problem (VRP). In this work, the suitability of using Grammatical Evolution to generate heuristics for the VRP was investigated.

Generating heuristics through policy matrix evolution

A new area of heuristic generation has emerged in recent years, focussing on the creation of policies to perform particular actions at a given point in a search. In general such methods create an 'index policy' [85], which assigns a score to each possible action at a given decision point, with the action given the largest score chosen.

Özcan and Parkes [172] introduced the notion of using policy matrices to solve combinatorial optimisation problems. In this paper, a Genetic Algorithm was used to search over a space of matrices which discretise the set of possible decisions when solving online bin packing problems. As this representation is general it is possible to formulate existing heuristics as matrices within this framework. Human-competitive performance was observed on a set of online bin packing benchmarks, with the evolved policy matrices outperforming standard heuristics from the literature such as first-fit and best-fit. This work was improved by Asta et al. [5] by reducing the search space of matrices that the Genetic Algorithm operates on. If the original representation is reduced effectively, it is possible for the Genetic Algorithm to find policies more quickly without sacrificing solution quality.

Parkes et al. [177] also investigated the evolution of policy matrices to solve the online bin packing problem, using Genetic Programming as a high-level search method. Although presented as a general study on the role of the mutation operator in Genetic Programming, this work proposes an interesting method of generating policy matrices from evolved Genetic Programming expressions. A policy matrix is created from the results of using all possible combinations of integer values as input for each individual in the population.

It is noted by Asta et al. [6] that despite the policy matrix approach of Özcan and Parkes [172] being able to generate heuristics which outperform existing heuristics, it is restricted to a specific set of bin capacities and range of item sizes. This paper investigates the scalability of a Genetic Algorithm-based policy matrix approach, training on a set of small instances before applying the policy matrices found to larger unseen instances. Rather than operating on problem-specific constraints, the evolved policy matrices cover a set of more general problem features. It is shown that results comparable to the method of Özcan and Parkes [172] and the traditional best-fit heuristic could be obtained for a set of online bin packing instances.

Recent techniques related to heuristic generation

Sim et al. [194, 195] used a method inspired by Artificial Immune Systems to manage sets of low-level heuristics to solve instances of the one-dimensional bin packing problem. The proposed system maintains a set of randomly generated heuristics, represented as trees, initiated using a standard construction method from the GP literature. Their results indicated that mixing heuristics in such a way was preferable to applying a single deterministic heuristic, over a large number of benchmark problem instances.

Although not explicitly concerned with generating heuristics, López-Ibáñez and Stützle [136, 137] introduced a framework to automatically 'construct' Ant Colony Optimisation algorithms (ACO) for multi-objective optimisation problems. This work described and generalised a number of components of ACOs, presenting the choice of which components to use as an algorithm configuration problem. By describing the design of ACOs as a set of component choices in such a way, it is possible to represent a large number of ACO variants, including some combinations of components not previously investigated. Using offline algorithm configuration techniques to select which components to use, this system was shown to outperform existing ACO methods on instances of the bi-objective travelling salesman problem. This work was extended by Bezerra et al. [13], who applied the automatically configurable ACO system to instances of the bi-objective bidimensional knapsack problem. Again, improved results were shown when compared to four existing ACO algorithms.

2.4 Recent advances in selection hyper-heuristics operating on perturbative low-level heuristic sets

In this section recent selection hyper-heuristic methods are covered, of which many are applied to multiple problem domains. A summary of the methods described here is provided in Appendix A. Table A.1 categorises selection hyper-heuristics by the heuristic selection method used, Table A.2 categorises selection hyper-heuristics by the acceptance criterion used and Table A.3 categorises selection hyper-heuristics by the problem domain solved.

Maturana et al. [149] used a variant of *Reinforcement Learning*, *Adaptive Operator Selection* [68], to select which crossover operator to use in evolutionary algorithms solving the satisfiability (SAT) problem. This work employed *Adaptive Operator Selection* over a subset of 20 'active' operators, updated at regular intervals throughout the search, from a total of 307 crossover operators split into four categories. Evolutionary algorithms using *Adaptive Operator Selection* over multiple crossover operators were shown to outperform using a single state-of-the-art crossover method in some benchmark instances of SAT.

García-Villoria et al. [77] applied a number of different hyper-heuristic methods to an NP-hard scheduling problem, the response time variability problem (RTVP). After introducing a number of constructive hyper-heuristics for the problem a set of single-point search hyper-heuristics were tested. *Simple Random*, *Greedy* and two variations of *Probability-based* heuristic selection were used to select a heuristic from a set of local search operators with *All Moves* accepted. The first variant of *Probability-based* heuristic selection used performance indicators to calculate the probability of selecting a given heuristic at each step. The second worked in much the same way however it contained a Tabu Search element, with a threshold used to exclude poor performing heuristics from selection as the search progresses. Using a hyper-heuristic to select a local search heuristic was shown to outperform naive iterative selection, with particular improvement shown by the *Probability-based* heuristic selection methods which include a learning mechanism. The local search heuristics were then replaced by a set of metaheuristics. The combination of metaheuristics within a hyper-heuristic framework was observed to outperform applying each of the metaheuristics individually.

A heuristic selection method modelled as a Markov chain was applied to the multi-objective DTLZ benchmark set by McClymont and Keedwell [150]. Essentially a *Reinforcement Learning* scheme, this heuristic selection mechanism maintains a set of weighted edges representing probabilities of transitioning from one heuristic to another. After each invocation, the edge weights are updated based on the heuristics performance. Given a set of four low-level heuristics, this heuristic selection method was incorporated into a Evolution Strategy framework and compared to *Simple Random* and a *Probability-based* heuristic selection method *'TSRoulWheel'*, proposed by Burke et al. [23]. A *Threshold-based* acceptance criterion which gradually increases the probability of accepting a non-improving move as the search stagnates, was combined with this heuristic selection method and applied to the benchmarks provided by HyFlex and submitted to CHeSC2011.

Kiraz et al. [124] and Kiraz et al. [126] tested a number of selection hyper-heuristics in dynamic environments. A suite of hyper-heuristics were tested using *Simple Random*, *Greedy*, *Random Permutation Descent*, *Choice Function* and *Reinforcement Learning* heuristic selection methods with *All Moves*, *Improving and Equal*, *Great Deluge* and a number of *Simulated Annealing* variants as acceptance criteria. Experiments were carried out using the Moving Peaks Benchmark to simulate landscapes with various different dynamic properties. Nine landscape properties were tested consisting of pairwise combinations of low, medium and high frequency of landscape change with low, medium and high severity of change. The selection hyper-heuristics operated over a set of seven low-level heuristics derived from a single Gaussian mutation operator, with seven discrete parameter values defining the level of mutation for each heuristic. Over all nine landscapes, *Choice Function - Improving and Equal* was shown to outperform all other hyper-heuristics.

The short-term electrical power generation scheduling (SEPGS) problem was solved using hyper-heuristics by Berberoglu and Uyar [12]. The objective of this problem is to minimise the cost of generating power whilst satisfying demand. As this problem can be represented using a binary encoding, seven generic mutation and hill climbing operators were taken from the Genetic Algorithms literature to use as low-level heuristics. Twenty-four *heuristic selection method - move acceptance criterion* combinations were tested using *Simple Random*, *Random Descent*, *Random Permutation*, *Random Permutation Descent*, *Greedy* and *Choice Function* heuristic selection with *All Moves*, *Only Improving*, *Improving and Equal* and *Great Deluge* move acceptance criteria. *Random Permutation Descent - Only Improving* performed best out of the twenty-four hyper-heuristics tested.

The ready-mixed concrete (RMC) delivery problem is a real-world combined scheduling and vehicle routing problem from the construction industry solved with hyper-heuristics by Misir et al. [153]. Using *Simple Random* heuristic selection, a new *Threshold-based* acceptance criterion was proposed. *'Adaptive Iteration Limited List-based Threshold Acceptance' (AILLA)* is a *Threshold-based* acceptance criterion where the threshold is set at the level of the previous 'best-of-run' before the current best-of-run was found. A list of previous best solutions is maintained, with the threshold increased to the level of the next most recent 'best-of-run' after a certain number of iterations. Working on a set of nine low-level heuristics, five hyper-heuristics were tested. *Simple Random* was combined with *AILLA*, *Late Acceptance Strategy*, *Simulated Annealing*, *Great Deluge* and *Improving and Equal* move acceptance criteria. Given ten minutes of execution time, the best performing hyper-heuristics were *Simple Random - AILLA* and *Simple Random - Late Acceptance Strategy*. When this was increased to an hour of execution time, *Simple Random - AILLA* began to clearly outperform *Simple Random - Late Acceptance Strategy*.

Cobos et al. [43] applied hyper-heuristics operating on a low-level set of metaheuristics for web document clustering. *Simple Random* and *Probability-based* 'Roulette Wheel' heuristic selection methods were used over a set of metaheuristics including Particle Swarm Optimisation, two Genetic Algorithms and a number of variants of Harmony Search, operating on a shared population of solutions. The hyper-heuristics were used to find the best metaheuristic to solve a number of benchmark datasets for text document clustering. The best metaheuristic identified by the hyper-heuristic was shown to outperform 'Carrot2', an existing web search performing web document clustering.

Bilgin et al. [15] applied a number of hyper-heuristics to the patient admission scheduling problem and the nurse rostering problem. These are two different examples of real-world scheduling problems arising in the area of healthcare. Using *Simple Ran-*

dom, *Choice Function* and a *Tabu Search* method which selects randomly between non-tabu heuristics, a number of individuals are generated at each step before the best is selected to be considered by the move acceptance criterion. Each of the three heuristic selection methods were used in conjunction with *Simulated Annealing*, *Great Deluge*, *Improving and Equal* and *Only Improving* move acceptance criteria. For the patient admission scheduling problem, hyper-heuristics using *Great Deluge* move acceptance criterion generally outperformed all other hyper-heuristics. For the nurse rostering instances *Choice Function - Great Deluge* obtained the best results of the hyper-heuristics tested.

Blazewicz et al. [17] investigated the use of selection hyper-heuristics to predict DNA sequences. A set of low-level heuristics were defined which manipulate a given sequence through insertion, deletion, swap and shift operations. The heuristic selection methods used were: the four variants of the *Choice Function* taken directly from Cowling et al. [47], a *Reinforcement Learning* scheme which also maintains a set of tabu heuristics and a mechanism which although termed 'Simulated Annealing', is described as a *Probability-based* heuristic selection method which selects heuristics based on the rankings of a *Reinforcement Learning* scheme. The *Choice Function* and *Reinforcement Learning* hyper-heuristics were tested with accept *All Moves* acceptance criterion. The *Probability-based* selection hyper-heuristic used *Simulated Annealing* as an acceptance criterion. Using a base set of six low-level heuristics, the *Probability-based heuristic selection - Simulated Annealing* hyper-heuristic outperformed all other hyper-heuristics. As an extension an updated set of fourteen low-level heuristics was also tested. In excess of seventy subsets of the extended set were used in conjunction with each of the six hyper-heuristics. In these tests promising results were shown by the 'Roulette Choice' variant of the *Choice Function* using a set of ten low-level heuristics.

A *Probability-based heuristic selection - Simulated Annealing* hyper-heuristic was applied to two different domains by Bai et al. [8]. The heuristic selection method used chooses a heuristic probabilistically based on the rankings of a Reinforcement Learning scheme with short-term memory. This hyper-heuristic demonstrated a high level of generality, performing well on instances of both bin packing and university course timetabling. Improved performance over an existing hyper-heuristic and some bespoke metaheuristics was shown for university course timetabling problems. This method was later adapted to solve the multimodal homecare scheduling problem [189], a combination of the vehicle routing problem with time windows and nurse rostering problem. Given an solution generated using Constraint Programming, a number of methods were tested to improve the quality of the initial solution. When compared with a number of techniques from the literature the hyper-heuristic showed satisfactory results.

A system which evolved sequences of low-level heuristics was presented by Ren et al. [188]. A Genetic Algorithm and Ant Colony Optimisation were both used to evolve a population of sequences of low-level heuristics to solve p-median problem. Each sequence of heuristics maintains an individual solution which it is applied to at each generation. Each hyper-heuristic operates over a number of low-level heuristics taken from the literature whose parameters are adaptively managed by a separate Ant Colony Optimisation scheme. Results are presented for a number of benchmarks of the p-median problem where it is shown that managing low-level heuristic parameters adaptively can outperform setting such parameters randomly.

Smet et al. [198] presented a generic model for nurse rostering problems, applying a suite of selection hyper-heuristics to a number of benchmark sets. Six hyper-heuristics were tested consisting of pairwise combinations of *Choice Function* or *Simple Random* heuristic selection and *Simulated Annealing*, *Great Deluge* or *Improving and Equal* acceptance criteria. Hyper-heuristics operated on a set of either four or five low-level heuristics depending on the instance being solved. Over the instances tested, *Choice Function - Simulated Annealing*, *Simple Random - Simulated Annealing* and *Simple Random - Great Deluge* offered the best performance with no statistically significant difference between these methods. Both hyper-heuristics using *Improving and Equal* acceptance criterion were found to perform poorly. The authors note that the acceptance criterion used has a more significant impact on the performance of a hyper-heuristic than heuristic selection mechanism, in line with the observations of Özcan et al. [174]. The hyper-heuristics were also compared to an Adaptive Large Neighbourhood Search (ALNS) method from the literature. In every instance, the best performing hyper-heuristic was found to match or improve the performance of ALNS.

A bi-level hyper-heuristic approach applied to black-box optimisation benchmarking (BBOB) functions was introduced by Krempser et al. [130]. At the highest level, a heuristic selection method is used to decide whether to use a Genetic Algorithm or Differential Evolution operator to update a shared population. Once this choice is made, another heuristic selection method is used to decide which operator to use within the individual metaheuristics. Three heuristic selection methods were used: *Simple Random*, *Adaptive Operator Selection* - a *Probability-based* heuristic selection method which uses the scores of a Reinforcement Learning scheme and a deterministic comparison based strategy known as '*fitness-based area under curve bandit*' (*AUC*) proposed by Fialho et al. [69]. The results show that using an intelligent selection strategy could outperform *Simple Random* selection at both levels. It is also observed that by splitting the levels of abstraction, first selecting which type of operator to use before selecting the specific operator, is beneficial when compared to considering a Genetic Algorithm or Differential Evolution hyper-heuristic independently.

Burke et al. [35] applied a number of hyper-heuristics to a set of examination timetabling benchmarks. Hyper-heuristics using either *Simple Random*, *Greedy*, *Choice Function* or *Reinforcement Learning* heuristic selection methods were tested in combination with three move acceptance criteria based on *Simulated Annealing*. The hyper-heuristics utilising *Reinforcement Learning* were shown to perform poorly in these experiments. Better performance was observed using *Simple Random* selection with the same move acceptance criterion. That an 'intelligent' mechanism is unable to learn which heuristic to apply at a given time suggests a complex relationship between heuristic selection method and move acceptance criteria.

The performance of a selection hyper-heuristic can be dependent on the parameters of the low-level heuristics chosen to solve a given problem. Misir et al. [156] applied a suite of hyper-heuristics using a set of eleven stochastic perurbative low-level heuristics to the patient admission scheduling problem. Each of the eleven heuristics used a parameter to specify the number of individuals to be produced by this heuristic at a given iteration. Once a heuristic selection method had chosen a heuristic to apply and the required number individuals was produced by the chosen heuristic, three strategies were used to decide which individual to retain and pass to the move acceptance criteria. A greedy strategy kept the best individual created, a first ascent strategy kept the first individual created which is an improvement over the original individual and a hill climbing strategy replaced the incumbent solution each time an improving individual was found until the required number of individuals was produced. In total, fourteen hyper-heuristics were testing using *Simple Random* and *Adaptive Dynamic Heuristic Set* [154] heuristic selection methods and *Simulated Annealing*, *Late Acceptance Strategy*, *Great Deluge*, *Improving and Equal*, *All Moves*, *Only Improving* and *Adaptive Iteration Limited List-based Threshold Acceptance* [153] move acceptance criteria. All hyper-heuristics were run for ten minute and fifty minute time periods. The results showed that the parameter settings of the low-level heuristics, in this case the number of individuals to be created at each iteration, the time limit imposed and the heuristic selection mechanism or move acceptance criterion used had the greatest effect of the quality of results for a given hyper-heuristic in different experimental conditions.

An online parameter tuning problem in a dynamic environment was studied by Köle et al. [128]. The Open Racing Car Simulator (TORCS) car setup optimisation problem involves searching for the best parameter settings of a race car to improve performance on a number of different tracks. A preliminary investigation involving six hyper-heuristics using *Simple Random*, *Random Descent* and *Reinforcement Learning* heuristic selection methods with *All Moves* and *Improving and Equal* acceptance criteria showed that the best performing hyper-heuristic in this environment was *Simple Random - Improving and Equal*. Operating over a set of eight mutation operators with fixed rates of mutation, this hyper-heuristic was compared to a number of population-based evolutionary algorithms from the literature. This hyper-heuristic outperformed both a Genetic Algorithm and Particle Swarm Optimisation however the application of a single heuristic from the low-level heuristic set was shown to offer slightly better performance.

A number of hyper-heuristics were used by Wauters et al. [211] to solve the Eternity II puzzle. The Eternity II puzzle belongs to an NP-complete class of edge matching puzzles. Hyper-heuristics with *Simple Random* heuristic selection were tested with *Improving and Equal*, *Great Deluge*, *Simulated Annealing*, *All Moves* and *'Iteration Limited Threshold Accepting' (ILTA)* move acceptance criteria. *Iteration Limited Threshold Accepting* is a *Threshold-based* move acceptance criterion which accepts non-improving solutions within a fixed range after a certain number of non-improving iterations. The five hyper-heuristics were tested over a set of three perturbative low-level heuristics and optimised over a number of different objective functions. It is shown that optimising with respect to a secondary objective function can lead to improved performance in a primary objective, rather than optimising directly over the primary objective function. The *Simple Random - ILTA* hyper-heuristic outperformed all other hyper-heuristics for this problem. A variant of this hyper-heuristic was the winner of the META'10 Eternity II contest.

Demeester et al. [56] used Simple Random-based hyper-heuristics for three exam timetabling datasets. *Improving or Equal*, *Great Deluge*, *Simulated Annealing*, *Late Acceptance Strategy* and *'Steepest Descent Late Acceptance'* were used as move acceptance criteria. *Steepest Descent Late Acceptance* is a variant of *Late Acceptance Strategy* which first applies the criteria for *Only Improving* to the candidate solution and the incumbent solution before applying the standard *Late Acceptance Strategy* criteria. This method is a slight variation of a traditional single-point search selection hyper-heuristic framework as at each step multiple candidate solutions are created. The best of these solutions is then passed to the move acceptance criterion to decide whether to accept the move. The *Simple Random - Simulated Annealing* hyper-heuristic improved on a number of best results from the literature over the Toronto benchmark dataset and performed well over a second dataset provided by the authors.

A *Reinforcement Learning* heuristic selection method analogous to Ant Colony Optimisation was proposed by Kiraz et al. [125]. In this system a pheromone value is maintained for each pair of heuristics $h_i h_j$, denoting the desirability of selecting h_j following the application of h_i. After each application of a low-level heuristic these values are updated according to the fitness value of the solution obtained. This is similar to the f_2 component of the *Choice Function* [47] (see Table 2.1) which seeks to utilise pair-wise synergy between low-level heuristics. Two variants of this heuristic selection method were proposed using standard methods taken from the evolutionary algorithms literature. The first selects a low-level heuristic using roulette wheel selection of all low-level heuristics, weighted by the pheromone levels of each heuristic pairing. The second uses tournament selection, tested for a variety of tournament sizes. These heuristic selection methods were shown to outperform the original *Choice Function* [47], *Reinforcement Learning* [162] and the *Modified Choice Function* [62] when combined with *Improving and Equal* move acceptance and applied to dynamic environment problems produced by the moving peaks benchmarks.

Misir et al. [151] presented a study analysing the effect of using different low-level heuristic sets in selection hyper-heuristics, over three real-world healthcare problem domains. This study focussed on a different flavour of generality than most hyper-heuristics studies. Rather than concentrating on the performance over differing problem domains, this study analysed the ability of hyper-heuristics to adapt to differing low-level heuristic sets. A large number of selection hyper-heuristics were tested using *Simple Random* and *ADHS* [154] heuristic selection with *AILLA* [153], *Great Deluge*, *Simulated Annealing*, *Late Acceptance Strategy*, *Improving and Equal*, *Only Improving* and *All Moves* acceptance criteria. The performance of the selection hyper-heuristics tested was observed to be linked to the low-level heuristics sets and termination criterion used. It is noted that if a hyper-heuristic is allowed a limited amount of execution time, ‘intelligent’ mechanisms for heuristic selection may not necessarily perform as well as simple ones. It is also observed that in the framework tested, a naive move acceptance criterion can be effective given the right low-level heuristic set.

2.5 The HyFlex framework

Hyper-heuristic research aims to ‘raise the level of generality’ at which search and optimisation methods operate. Recently there has been increased interest in cross-domain optimisation, i.e. methods which are able to perform well over more than one problem domain. This is due in part to the introduction of HyFlex [27, 166], a multi-domain framework designed to aid research into heuristic search methods. Rather than having to develop and tune methods to individual problems or even instances of problems, cross-domain optimisation methods attempt to be general enough to provide an acceptable level of performance, irrespective of the underlying problem domain they are solving.

The HyFlex [27, 166] framework[1] offers a common framework in which to test heuristic search algorithms over a broad set of problem domains. This framework was used to support an international research competition, the first Cross-domain Heuristic Search Challenge (CHeSC2011) [31]. Before the competition four problem domains were provided: Boolean satisfiability (MAX-SAT), one-dimensional bin packing, personnel scheduling and permutation flow shop. Following the competition an extra two problem domains, the vehicle routing problem (VRP) and the travelling salesman problem (TSP), were made available. For each problem domain a set of low-level heuristics are defined and categorised as either 'ruin-recreate', 'mutation', 'local search' or 'crossover'. Each of these problem domains is discussed in detail in Section 2.5.2.

A summary of the work discussed in the following section is given by heuristic selection method and move acceptance criteria in Table A.4 and Table A.5 respectively in Appendix A.

2.5.1 Recent studies using the CHeSC2011 benchmark problems

Burke et al. [32] embedded two heuristic selection methods into an Iterated Local Search [140, 141] framework. Iterated Local Search typically contains two phases, a diversification phase and an intensification phase. During the diversification phase, a perturbation is made to the current solution to move into a different region of the search space. Following this in the intensification phase, local search is performed to reach the local optima of the current area of the search space. This process is repeated until a given termination criterion is met. The *Choice Function* and a *Reinforcement Learning* scheme known as *'Extreme value-based Adaptive Operator Selection'* were compared to a baseline *Simple Random* selection strategy, as heuristic selection methods used in an Iterated Local Search framework. Following the selection and application of a heuristic the Iterated Local Search framework applied each available local search heuristic to the candidate solution, keeping the individual which yielded the greatest improvement. Using the best individual resulting from the application of local search, *Only Improving* moves when compared to the original solution were accepted by the algorithm. The three Iterated Local Search variants were applied to four different problem domains

[1]Available at: `http://www.asap.cs.nott.ac.uk/external/chesc2011/index.html`

using the HyFlex framework. The *Reinforcement Learning* scheme performed well on three of the four problem domains, matching and improving some best-known solutions for the personnel scheduling problem. On the fourth problem domain, permutation flow shop, the *Choice Function* outperformed both other variants in all instances. An extension to this method was presented by Walker et al. [209] showing improved performance over instances of the vehicle routing problem. The adapted algorithm replaced the greedy application of local search operators with an ordered application scheme based on previous performance.

Genetic Programming has often been used in the field of hyper-heuristics as a method of generating heuristics or heuristic components [28] (see Section 2.3.4). Nguyen et al. [164] used Genetic Programming to evolve disposable selection hyper-heuristics to solve individual instances of three of the HyFlex benchmark problem domains: MAX-SAT, bin packing and permutation flow shop. The Genetic Programming system used a function set containing a number of conditional operators and acceptance criteria (*All Moves*, *Only Improving* and *Simulated Annealing*) and a terminal set consisting of the low-level heuristics provided for each problem domain. This method showed improved results compared to human-designed hyper-heuristics albeit at a much greater computational cost, as each instance requires a full Genetic Programming run. Although there is an extra overhead of computational cost in generating hyper-heuristics in this manner, the automation of the heuristic design process is an important goal of hyper-heuristic research.

Misir et al. [154] introduced a hyper-heuristic based on a new heuristic selection method and *Adaptive Iteration Limited List-based Threshold Acceptance (AILLA)* [153] acceptance criterion. The heuristic selection method, *'Adaptive Dynamic Heuristic Set' (ADHS)*, is an intelligent mechanism which maintains an set of tabu heuristics made unavailable for selection based on a number of metrics gathered during the search process. At each stage a poor performing heuristic is applied, before a *Probability-based* heuristic selection method selects a second heuristic to apply. These components were shown to work well with each other when applied to four HyFlex benchmarks; MAX-SAT, bin packing, personnel scheduling and permutation flow shop. *ADHS - AILLA* outperformed *Simple Random - AILLA*, *Simple Random - Simulated Annealing* and *Simple Random - Only Improving* in most of the instances of the MAX-SAT, bin packing and personnel scheduling problem domains. In the permutation flow shop domain, *Simple Random - Simulated Annealing* outperformed the other three hyper-heuristics in all ten problem instances. *ADHS - AILLA* was later submitted to CHeSC2011 [155] outperforming all other competition entrants in MAX-SAT, bin packing and the travelling salesman problem. It was also classified as the overall winner of CHeSC2011 based on the competition scoring method.

A heuristic selection method based on the idea of dominance was presented by Özcan and Kheiri [171] and applied to the first four problem domains of the HyFlex benchmark set: MAX-SAT, bin packing, personnel scheduling and permutation flow shop. A greedy phase repeatedly applies each heuristic to a solution for a given number of steps. Heuristics which are dominated by another heuristic at every point are placed on a tabu list. Once the greedy phase is complete, the non-tabu heuristics are selected and applied using the *Random Descent* heuristic selection method [47]. A naive move acceptance criterion which accepts all improving moves is used at each step in this phase. In the case of non-improving moves the move is accepted with probability 0.25, the move is rejected and the *Random Descent* phase continued with probability 0.5 and the move is rejected and the algorithm returned to the greedy phase with probability 0.25. This hyper-heuristic outperformed a set of eight 'default hyper-heuristics' provided by the CHeSC2011 organisers, performing particularly well in the MAX-SAT and bin packing domains. The default hyper-heuristics were based on state-of-the-art methods from the literature and provided by the HyFlex framework.

Chan et al. [39] developed an Iterated Local Search-based hyper-heuristic inspired by real-world pearl hunting, a diving technique used to retrieve pearls from pearl oysters. This scheme repeatedly applies a low-level heuristic to move the search to different areas of search space before using local search to try to improve the solution. A static threshold, set at the level of the best result achieved during the first iteration, is used to decide whether to accept the initial move in the search space. If the quality of solution does not meet this threshold, local search is not applied and a diversification operator is applied again to reach a new point in the search space. This hyper-heuristic was tested on the original four problem domains of HyFlex and submitted to CHeSC2011. The perturbative low-level heuristics used to diversify the search were applied in an arbitrary order during the first half of the available computation time. During the second half of available computation time, low-level heuristics were applied in order of performance during the first half of the run. The local search phase repeatedly applied sequences of the available local search heuristics to the current solution. This hyper-heuristic finished fourth out of twenty entries to CHeSC2011, outperforming all other competition entrants in the vehicle routing problem domain.

Another hyper-heuristic based on an Iterated Local Search framework was presented by Lehrbaum and Musliu [133]. The diversification phase applies a perturbative operator, chosen by a *Probability-based* heuristic selection method, based on relative previous performance. The improvement phase applies each available heuristic in descending order of previous performance, accepting solutions which are *Improving and Equal* to the current solution. In the case of equal quality solutions, only solutions which differ from the current solution are accepted. A tabu list of recently seen solutions is also maintained to prevent the search from returning to these points. Persistently poor performing low-level heuristics are temporarily placed on a tabu list with a given probability at each step of the search process. This hyper-heuristic performed particularly well on instances of MAX-SAT and personnel scheduling and finished sixth out of twenty entries to CHeSC2011.

Ochoa et al. [167] compared a modified version of the Adaptive Iterated Local Search (AILS) algorithm presented by Walker et al. [209] with an Adaptive Memetic Algorithm. The Adaptive Memetic Algorithm was effectively a parallel Iterated Local Search with four threads, periodically recombined through the use of crossover in order to share information. The Adaptive Memetic Algorithm was shown to outperform the best hyper-heuristics submitted to CHeSC2011 on the vehicle routing problems contained in HyFlex.

Hsiao et al. [105] proposed a population-based hyper-heuristic set within an Iterated Local Search framework. At each step of the first half of available computation time, an individual is selected from a small population using the tournament selection strategy and a perturbative low-level heuristic is applied to modify the solution. All available local search operators are then repeatedly applied until no further improvement is possible. In the second half of available computation time rather than using tournament selection, the population is reduced to a single solution consisting of the best individual found so far during the run. The perturbative heuristics are selected in ascending order of perceived severity in a cyclic manner before the local search operators are applied. If an improving solution is found following the local search phase, the same perturbative heuristic is chosen in the next iteration and the improved solution replaces the incumbent solution in the population. If a solution of equal quality is generated, the decision of which perturbative heuristic to apply is performed probabilistically, with the new solution replacing the incumbent solution in the population. In the case a non-improving individual is produced following the local search phase, the next low-level heuristic is used following the order of severity and the worst individual in the population is replaced by the produced individual. This hyper-heuristic was applied to the first four HyFlex benchmark domains in this paper however results for all six domains are available as this method was submitted to CHeSC2011. When compared to the other twenty competitors, this method finished second using the competition scoring system. This hyper-heuristic was first in the personnel scheduling problem domain and good performance was shown on MAX-SAT, travelling salesman problem and permutation flow shop instances.

Three variants of *Reinforcement Learning* heuristic selection methods were presented by Gaspero and Urli [81], using techniques taken from the machine learning literature. The first is a traditional reward-based *Reinforcement Learning* scheme. The second controls the parameters of *Reinforcement Learning* through the use of an Artificial Neural Network (ANN) [93]. The final variant rewards not only the heuristic applied at each step, but also each of the preceding heuristics at a specified rate of degradation. Over the six problem domains used in the CHeSC2011 competition, the *Reinforcement Learning* heuristic selection mechanism utilising ANNs outperformed the other two variants. Little detail on performance compared to other techniques in general or in particular problem domains is given.

Cichowicz et al. [41] introduced two hyper-heuristics, a *'Five Phase'* hyper-heuristic and a *'Genetic Hive'* hyper-heuristic. The *Genetic Hive* hyper-heuristic evolved sequences of low-level heuristics to apply during the search process using the Bees Algorithm [179]. The *Five Phase* hyper-heuristic ran a number of parallel streams, sharing low-level heuristic performance information between streams. Each individual stream consists of a phase of intensification followed by a phase of diversification, repeated until no improvement is made in solution quality. Following this the search then continues in the next stream. If no improvement is made in three successive iterations of all streams, crossover operators are used to recombine solutions from different streams and reinitialise the algorithm. Using a number of different parameter settings for these two hyper-heuristics a total of fourteen hyper-heuristics are compared, including standard hyper-heuristics such as *Simple Random - All Moves* and *Greedy - All Moves* using the HyFlex benchmark problem domains. The *Genetic Hive* hyper-heuristic was submitted to CHeSC2011 and finished 12th out of the 20 competition entries.

Kubalik [131] used an evolutionary algorithm (EA) to evolve sequences of low-level heuristics to apply to a solution at a given point of the search. A design choice was made to impose certain structures on each sequence such that the first and last heuristic in a given sequence is a local search heuristic. At each step of the search, a permutation of low-level heuristics is applied in the order specified by a sequence in the EA with *All Moves* accepted. This method performed well on the one-dimensional bin packing instances in HyFlex, and finished eighth overall in the CHeSC2011 competition. Improved results were shown for a number of problem domains when some parameters were tuned offline and an adaptive reinitialisation strategy was included.

The hyper-heuristic set parameters for each problem domain independently, based on the execution time of the low-level heuristics in the given domain. Effectively, tuning the parameters in an offline manner in this way breaks the domain barrier at which a hyper-heuristic operates leading to a lower level of generality for this method.

Mascia and Stützle [147] developed a number of automatically tuned hyper-heuristics, each tuned to solve one of the first four HyFlex problem domains independently, using an Iterated Racing procedure [138]. Based on a method which finished seventh overall in CHeSC2011, these hyper-heuristics used a number of methods from the literature including *Iterated Local Search*, *Simulated Annealing* and *Greedy* and *Simple Random* heuristic selection methods. The tuned methods were then included in a high-level framework which selects a hyper-heuristic to use on a given problem instance. The iterated racing procedure is applied again to decide which of the hyper-heuristics to use for a given problem. The original hyper-heuristic performed well on bin packing and permutation flow shop instances. This work showed improvement over the original hyper-heuristic could be made through additional extensive offline tuning, again at the cost of decreased generality.

Kalender et al. [116, 117] applied a *Reinforcement Learning - Simulated Annealing* hyper-heuristic to university course timetabling problems. Each of the low-level heuristics implemented for this problem act as neighbourhood operators working on a single constraint. The so-called *Greedy Gradient* heuristic selection method uses a *Reinforcement Learning* strategy until non-improving moves are made by the last invocation of all low-level heuristics. When the search process stagnates in such a way, the heuristic selection method is changed and *Greedy* heuristic selection is used instead. In the case that no low-level heuristic offers an improvement in solution quality, the low-level heuristic which operates on the constraint which is most violated is chosen. Due to this feature, this method operates both above the domain barrier at the hyper-heuristic level and below domain barrier at the problem-specific level. The *Greedy Gradient* heuristic selection method was shown to outperform *Choice Function*, *Simple Random* and *Greedy* heuristic selection methods when combined with *Simulated Annealing* acceptance criterion on instances of the Yeditepe University course timetabling problem. The improvement in performance was observed to be greater as the size of the instances increased. *Reinforcement Learning - Simulated Annealing* was then also applied to all six benchmark domains found in HyFlex performing well on instances of the personnel scheduling and permutation flow shop domains.

Kheiri and Özcan [122] introduced a *'Round-robin'* heuristic selection mechanism for solving the HyFlex benchmark instances. This heuristic selection method is a variation of the *Random Permutation Gradient* heuristic selection method presented in the early hyper-heuristic work of Cowling et al. [47]. Both heuristic selection mechanisms apply low-level heuristics in a randomly generated order. However, where the original *Random Permutation Gradient* selection repeatedly applied low-level heuristics to a given solution until no improvement in fitness is observed, the *Round-robin* heuristic selection mechanism applies each low-level heuristic for a specified period of time. This heuristic selection method is tested in combination with a *Threshold-based* acceptance criterion, which accepts all improving moves and gradually increases the probability of accepting a non-improving move each time such a move is made. This hyper-heuristic was shown to work well on the six benchmark problem domains provided by HyFlex, outperforming all competition entrants in the vehicle routing problem domain and also scoring well in the MAX-SAT and bin packing domains.

Jackson et al. [112] compare a number of *Late Acceptance Strategy*-based selection hyper-heuristics over the HyFlex benchmark instances. A number of variants of a *Probability-based* heuristic selection mechanism inspired by the Formula One ranking system used in CHeSC2011 are compared to *Simple Random* heuristic selection when coupled with *Late Acceptance Strategy* move acceptance. Using a *Simple Random - Late Acceptance Strategy* hyper-heuristic was shown to outperform a number of the CHeSC2011 competitors when using the competition ranking method. The best results for the Formula One ranking-inspired *Probability-based* heuristic selection mechanism were in the personnel scheduling and MAX-SAT problem domains.

2.5.2 Problem domains used in the CHeSC2011 competition

The HyFlex [27, 166] framework was developed to support research in cross-domain optimisation and heuristic search. As one of the fundamental goals of hyper-heuristics is to raise the level of generality at which search methods operate, it is necessary to test new methods on a wide variety of problem domains. For CHeSC2011 six problem domains with an associated set of low-level heuristics were provided through the HyFlex framework. The following subsections describe each problem domain in turn, including the instances available and the set of low-level heuristics provided. A summary of the number of low-level heuristics and problem instances provided by each problem domain is given in Table 2.4.

Table 2.4: Summary of the number of instances and low-level heuristics provided for each problem domain in HyFlex

Domain Name	Instances	Ruin-recreate	Mutation	Local Search	Crossover
Bin Packing	12	2	3	2	1
MAX-SAT	12	1	6	2	2
Permutation Flow Shop	12	2	5	4	4
Personnel Scheduling	12	3	1	5	3
TSP	10	1	5	3	4
VRP	10	2	3	3	2

One-Dimensional Bin Packing

The one-dimensional bin packing problem consists of a set of items to be packed into a homogeneous set of containers (bins). Each item j has an associated weight w_j and each container has capacity c. The objective is to assign each item to a bin, minimising the number of bins needed to pack all of the items, whilst ensuring the sum of the weight of the pieces in each bin does not exceed the bin capacity. This problem was formulated by Martello and Toth [146] as follows:

$$\begin{aligned}
&\text{minimise} && \sum_{i=1}^{n} y_i && && (2.5.1)\\
&\text{subject to} && \sum_{j=1}^{n} w_j x_{ij} \le c y_i, && i \in N = \{1, ..., n\}, && (2.5.2)\\
& && \sum_{i=1}^{n} x_{ij} = 1, && j \in N && (2.5.3)\\
&\text{with} && y_i \in \{0,1\}, && i \in N, && (2.5.4)\\
& && x_{ij} \in \{0,1\}, && i \in N, j \in N && (2.5.5)
\end{aligned}$$

where y_i is a binary variable indicating whether bin i is used, x_{ij} denotes whether item j is packed into bin i and n is the total number of available bins. The one-dimensional bin packing problem has roots in a number of practical application domains such as cutting lengths of stock material [197] and scheduling advertising slots [1].

Rather than simply counting the number of bins used to measure the quality of a solution the fitness is calculated as:

$$fitess = 1 - \left(\frac{\sum_{i=1}^{n} (fullness_i / C)^2}{n} \right) \qquad (2.5.6)$$

where n is the number of bins, C is the capacity of each bin and $fullness_i$ is the total weight of all items in bin i. This function gives a value between zero and one with a set of completely full bins corresponding to a value of zero. Minimising this function corresponds to the overall objective of minimising the number of bins used n.

Twelve different problem instances are provided from a number of sources from the literature [67, 108]. The problem instances chosen vary widely in terms of the distribution of item sizes, the number of items to be packed and the capacity of the bins. These are summarised in Table 2.5. The low-level heuristic set contains two ruin-recreate heuristics, three mutation heuristics, two local search heuristics and one crossover-heuristic.

Table 2.5: Summary of the one-dimensional bin packing instances included in HyFlex

Index	Instance Name	Source	Capacity	Items
0	falkenauer/u1000-00	[67]	150	1000
1	falkenauer/u1000-01	[67]	150	1000
2	schoenfieldhard/BPP14	[67]	1000	160
3	schoenfieldhard/BPP832	[67]	1000	160
4	10-30/instance1	[108]	150	2000
5	10-30/instance2	[108]	150	2000
6	triples1002/instance1	[108]	1000	1002
7	triples2004/instance1	[108]	1000	2004
8	test/testdual4/binpack0	[67]	100	5000
9	test/testdual7/binpack0	[67]	100	5000
10	50-90/instance1	[108]	150	2000
11	test/testdual10/binpack0	[67]	100	5000

Boolean Satisfiability (MAX-SAT)

The Boolean satisfiability problem is a well-studied combinatorial optimisation problem, whereby an assignment of a set of binary variables of a Boolean formula is sought such that the formula evaluates to true. As an example, Equation 2.5.7 is satisfied if the variables are assigned as follows: $x_1 = true$, $x_2 = true$, $x_3 = true$, $x_4 = true$.

$$(x_1 \vee \neg x_2 \vee \neg x_3) \ \wedge \ (x_1 \vee x_2 \vee x_4) \tag{2.5.7}$$

The maximum satisfiability problem (MAX-SAT) is a variant of the satisfiability problem for which the given Boolean formula is not necessarily 'satisfiable'. In MAX-SAT, the objective is to maximise the number of clauses satisfied in the Boolean formula.

This problem is modelled as a minimisation problem in HyFlex where the objective is to minimise the number of unsatisfied clauses.

The instances available are taken from a number of sources in the literature [50, 51, 4]. These are summarised in Table 2.6. The low-level heuristic set contains one ruin-recreate heuristic, six mutation heuristics, two local search heuristics and two crossover-heuristics.

Table 2.6: Summary of the Boolean satisfiability (MAX-SAT) instances included in HyFlex

Index	Instance Name	Source	Variables	Clauses
0	contest02-Mat26.sat05-457.reshuffled-07	[50]	744	2464
1	hidden-k3-s0-r5-n700-01-S2069048075.sat05-488.reshuffled-07	[50]	700	3500
2	hidden-k3-s0-r5-n700-02-S350203913.sat05-486.reshuffled-07	[50]	700	3500
3	parity-games/instance-n3-i3-pp	[51]	525	2276
4	parity-games/instance-n3-i3-pp-ci-ce	[51]	525	2336
5	parity-games/instance-n3-i4-pp-ci-ce	[51]	696	3122
6	highgirth/3SAT/HG-3SAT-V250-C1000-1	[4]	250	1000
7	highgirth/3SAT/HG-3SAT-V250-C1000-2	[4]	250	1000
8	highgirth/3SAT/HG-3SAT-V300-C1200-2	[4]	300	1200
9	MAXCUT/SPINGLASS/t7pm3-9999	[4]	343	2058
10	jarvisalo/eq.atree.braun.8.unsat	[50]	684	2300
11	highgirth/3SAT/HG-3SAT-V300-C1200-4	[4]	300	1200

Permutation Flow Shop

The permutation flow shop scheduling problem emerged from real-world problems in the manufacturing industry. Given a set of jobs n and a set of machines m, the objective is to minimise the completion time of the last job to leave the system. Each job i has an associated processing time on machine j denoted by p_{ij}.

All of the benchmarks used for this problem domain are provided by Taillard [202] and are summarised in Table 2.7. The low-level heuristic set contains two ruin-recreate heuristics, five mutation heuristics, four local search heuristics and four crossover-heuristics.

Table 2.7: Summary of the permutation flow shop instances included in HyFlex

Index	Instance Name	Source	Jobs	Machines
0	10x20/1	[202]	100	20
1	10x20/2	[202]	100	20
2	10x20/3	[202]	100	20
3	10x20/4	[202]	100	20
4	10x20/5	[202]	100	20
5	200x10/2	[202]	200	10
6	200x10/3	[202]	200	10
7	500x20/1	[202]	500	20
8	500x20/2	[202]	500	20
9	500x20/4	[202]	500	20
10	200x20/1	[202]	200	20
11	500x20/3	[202]	500	20

Personnel Scheduling

The personnel scheduling problem involves allocating the days and times at which a set of employees should work over a given planning period. In particular, the problems included in HyFlex are taken from the nurse rostering domain. A weighted objective function is defined to measure the quality of a solution based on a number of constraints and objectives. As an example, certain shift patterns may be undesirable or not allowed (i.e. allocating a single staff member to work every day of the planning period). There may also be shift cover requirements, whereby the set of staff on duty must possess a certain set of skills.

The problem instances used are taken from two sources [53, 111] with the vast majority drawn from real-world personnel scheduling examples. A set of low-level heuristics is provided containing three ruin-recreate heuristics, one mutation heuristic, five local search heuristics and three crossover heuristics.

Table 2.8: Summary of the personnel scheduling instances included in HyFlex

Index	Instance Name	Source	Staff	Shift Types	Days
0	BCV-3.46.1	[53]	46	3	26
1	BCV-A.12.2	[53]	12	5	31
2	ORTEC02	[53]	16	4	31
3	Ikegami-3Shift-DATA1	[111]	25	3	30
4	Ikegami-3Shift-DATA1.1	[111]	25	3	30
5	Ikegami-3Shift-DATA1.2	[111]	25	3	30
6	CHILD-A2	[53]	41	5	42
7	ERRVH-A	[53]	51	8	42
8	ERRVH-B	[53]	51	8	42
9	MER-A	[53]	54	12	42
10	BCV-A.12.1	[53]	12	5	31
11	ORTEC01	[53]	16	4	31

Travelling Salesman Problem (TSP)

The travelling salesman problem is one of the most well-known and well-studied problem domains in combinatorial optimisation. Given a set of cities and the associated travel costs between each pair of cities, the objective of the travelling salesman problem is to find the cheapest route, visiting every city once and returning to the starting city. Although originally formulated in the 1930's, the travelling salesman problem still has a number of relevant real-world applications such as planning, logistics and microchip manufacture.

Ten instances are provided for this problem domain taken from TSPLIB [187] and are summarised in Table 2.9. The number of each type of low-level heuristic implemented are as follows: one ruin-recreate heuristic, five mutation heuristics, three local search heuristics and two crossover-heuristics.

Table 2.9: Summary of the travelling salesman problem instances included in HyFlex

Index	Instance Name	Source	Cities
0	pr299	[187]	299
1	pr439	[187]	439
2	rat575	[187]	575
3	u724	[187]	724
4	rat783	[187]	783
5	pcb1173	[187]	1173
6	d1291	[187]	1291
7	u2152	[187]	2152
8	usa13509	[187]	13509
9	d18512	[187]	18512

Vehicle Routing Problem (VRP)

The vehicle routing problem (VRP) is an NP-Complete [78] combinatorial optimisation problem where a number of customers are to be serviced by a fleet of vehicles subject to a number of constraints. Different objectives can be considered depending on the goal of the problem. Typical objectives include; minimisation of cost with respect to distance travelled, minimisation of the global travel time, minimisation of the number of vehicles required to service all customers, minimisation of the penalty costs associated with partial service of customers. The objective could also be a weighted combination of multiple objectives. Real-world commodity distribution in logistics is a complex problem with constraints varying depending on the application. It is therefore natural that many different variants of the VRP exist, each simplifying the problem to a smaller set of constraints which impose the most important restrictions for each specific application of the problem. A large number of exact [132, 204] and metaheuristic [44, 19] methods have been applied in the literature to solve vehicle routing problems.

A total of ten instances are provided for this problem domain taken from [196] and are summarised in Table 2.10. Two ruin-recreate, three mutation, three local search and two crossover low-level heuristics are implemented.

Table 2.10: Summary of the vehicle routing problem instances included in HyFlex

Index	Instance Name	Source	Vehicles	Capacity	Customers
0	RC207	[196]	25	1000	100
1	R101	[196]	25	200	100
2	RC103	[196]	25	200	100
3	R201	[196]	25	1000	100
4	R106	[196]	25	200	100
5	C1-10-1	[196]	250	200	1000
6	RC2-10-1	[196]	250	1000	1000
7	R1-10-1	[196]	250	200	1000
8	C1-10-8	[196]	250	200	1000
9	RC1-10-5	[196]	250	200	1000

2.5.3 CHeSC2011 results and scoring system

In order to compare the hyper-heuristics submitted to the competition, CHeSC2011 used a points-based scoring system inspired by a system previously used by Formula One motor racing to rank performance. The Formula One ranking system (2003-2009) assigns a number of points to different competitors based on their performance. The first place competitor is awarded 10 points, the second 8 points and then each further hyper-heuristic is awarded 6, 5, 4, 3, 2, 1 and 0 points respectively. As the Formula One ranking system allocates scores to the top 8 ranked contestants, all hyper-heuristics which are ranked $\geq$ 9th position are given a score of 0.

A selection of five instances were selected from each of the six problem domains to use as testing instances, resulting in a total of 30 instances. The HyFlex indexes of the instances used in the competition are summarised in Table 2.11. Of these instances, twenty belong to problem domains which were available to all contestants prior to the competition (Boolean satisfiability (MAX-SAT), one-dimensional bin packing, personnel scheduling and permutation flow shop). A further ten, five from the vehicle routing problem and five from the travelling salesman problem were unseen problems. Each hyper-heuristic was allowed to run for 10 minutes per instance on a standard desktop machine, with runs repeated 31 times in order to account for the stochastic nature of solving such optimisation problems. The median result of the 31 runs is reported as the score for a given hyper-heuristic applied to a given instance. Scores are then allocated using the Formula One system outlined above, using the median results obtained by each hyper-heuristic for each instance. As the maximum number of points for each instance is 10, and there are 30 instances in total, the maximum possible score for any

Table 2.11: HyFlex indexes of problem instances used for the CHeSC2011 competition

	Domain Name	Instances
Seen	Bin Packing	7, 1, 9, 10, 11
	MAX-SAT	3, 5, 4, 10, 11
	Permutation Flow Shop	1, 8, 3, 10, 11
	Personnel Scheduling	5, 9, 8, 10, 11
Unseen	TSP	0, 8, 2, 7, 6
	VRP	6, 2, 5, 1, 9

hyper-heuristic is 300.

In total there were twenty entrants to CHeSC2011 as shown in Table 2.12. These are listed in order of Formula One ranking as defined by the competition rules. References to the methods submitted to the competition are provided where available.

Table 2.12: Results of CHeSC2011 using the Formula One ranking system

Rank	Name	Score	Reference
1	AdapHH	181.00	Misir et al. [155]
2	VNS-TW	134.00	Hsiao et al. [105]
3	ML	131.50	-
4	PHunter	93.25	Chan et al. [39]
5	EPH	89.75	-
6	HAHA	75.75	Lehrbaum and Musliu [133]
7	NAHH	75.00	Mascia and Stützle [147]
8	ISEA	71.00	Kubalik [131]
9	KSATS-HH	66.50	-
10	HAEA	53.50	-
11	ACO-HH	39.00	-
12	GenHive	36.50	Cichowicz et al. [41]
13	DynILS	27.00	-
14	SA-ILS	24.25	-
15	XCJ	22.50	-
16	AVEG-Nep	21.00	Gaspero and Urli [81]
17	GISS	16.75	-
18	SelfSearch	7.00	-
19	MCHH-S	4.75	McClymont and Keedwell [150]
20	Ant-Q	0.00	-

2.6 Concluding remarks

This chapter introduced much of the existing related work in the literature and serves to contextualise the work contained in the thesis herein. Previous and recent work in hyper-heuristics, specifically selection hyper-heuristics and the HyFlex framework have been discussed. Hyper-heuristics have been defined as a class of high-level search methodologies which operate on a search space of low-level heuristics or heuristic components. Hyper-heuristic research has a number of goals, chiefly to raise the generality at which search methods operate and to provide satisfactory results to a given problem within a reasonable period of time.

The introduction of the HyFlex framework has greatly improved the ease in which high-level search strategies can be tested over multiple problem domains. A number of selection hyper-heuristics designed specifically to perform cross-domain optimisation have been detailed. The HyFlex framework includes a number of low-level heuristic types for high-level search methods to operate on including crossover heuristics. Crossover operators require > 1 solutions as input however there is little work in the literature devoted to understanding how to select and manage these solutions. In fact many of the state-of-the-art methods using this benchmark avoid this issue by omitting crossover operators altogether [105, 133, 147]. In the worst case whereby only crossover operators are available, it is possible that these methods will fail entirely. This problem is not restricted to crossover operators and can be generalised to all n-ary operators where n is the number of solutions required as input by a low-level heuristic, assuming $n > 1$. As there are some domains where crossover use is provably beneficial, it does not make sense to completely eliminate this category of low-level heuristic from the search space.

In this thesis the management of a number of high-level selection hyper-heuristic components is explored, particularly the management of multi-input low-level heuristics. The following chapter introduces the multidimensional knapsack problem, a domain which will be used as a case study to compare some of the methods presented. The multidimensional knapsack problem is a widely studied combinatorial optimisation problem from the literature, for which a large number of benchmark problems exist. Selection hyper-heuristics consisting of well-known existing components are tested on this problem domain, showing some success when compared to existing methods.

CHAPTER 3

The Multidimensional Knapsack Problem

3.1 Introduction

The previous chapter presented a review of the relevant literature regarding hyper-heuristics. A number of problem domains included in the HyFlex benchmark were introduced. This chapter will provide a brief introduction to another problem domain, the multidimensional knapsack problem (MKP). In subsequent chapters this problem domain will be used as an additional case study to compare different selection hyper-heuristics.

The multidimensional knapsack problem was chosen as a benchmark for a number of reasons. Firstly there are a number of existing benchmark datasets containing problem instances with varying properties. Each of these benchmark sets includes a large number of problem instances. As a core goal of hyper-heuristic research is to 'raise the level of generality' at which search methods operate [190], a large, diverse set of problem instances and problem instance types is beneficial to measure performance over a variety of experimental conditions. Secondly there is a vast body of existing work in the literature solving this problem, including both exact and metaheuristic methods. This provides a broad set of techniques with which to compare the hyper-heuristics developed. Finally, to best of the authors knowledge, this is the first application of selection hyper-heuristics to this problem domain, offering a previously unexplored research area.

A formal definition of the multidimensional knapsack problem is provided, along with a review of previous work applied to this domain. A description of the problem instances contained within three well-known benchmark datasets is also included.

3.2 Problem definition

The NP-hard [78] multidimensional 0-1 knapsack problem (MKP) [212] is a generalised case of the standard 0-1 knapsack problem, with roots in applications such as capital budgeting [139] and project selection [178]. The MKP is a resource allocation model, where the objective is to select a subset of objects which yield the greatest profit, whilst observing the constraints on knapsack capacities. Unlike the standard 0-1 knapsack problem, each object j consumes a different amount of resources in each dimension i when selected.

Formally the MKP can be stated as:

$$\text{maximise} \quad \sum_{j=1}^{n} p_j x_j \tag{3.2.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad i = 1, ..., m \tag{3.2.2}$$

$$\text{with} \quad x_j \in \{0, 1\}, \qquad j = 1, ..., n \tag{3.2.3}$$

where p_j is the profit for selecting item j, a_{ij} is the resource consumption of item j in dimension i, b_i is the capacity constraint of each dimension i. Using direct binary encoding, x_1,...,x_n is a set of decision variables indicating whether or not object j is included in the knapsack. The size of a problem is defined by the total number of variables n and the number of dimensions m. Tavares et al. [203] investigated five different representations and analysed their effect on solution quality. This work highlighted that using direct binary encoding in conjunction with local search or repair operators in both mutation-based and crossover-based evolutionary algorithms is suitable for the MKP.

3.2.1 Fitness function

In any optimisation problem a measure is needed to assess the quality of a solution. There are a number of options when choosing a fitness function for the MKP. In this thesis the following fitness function taken from [170] is used:

$$profit - o * s * (maxProfit + 1) \tag{3.2.4}$$

where $profit$ is the profit gained from the items currently selected for inclusion, o is the number of dimensions for which resource consumption has exceeded capacity, s is the number of selected items and $maxProfit$ is the largest profit value of any of the items. The value of this fitness function will always be positive for a feasible solution and negative for an infeasible solution.

3.3 Literature review

A number of methods, including exact and metaheuristic techniques, have been used to solve the MKP and its single constraint equivalent. Drexl [64] used Simulated Annealing [127] within a variable exchange operator to solve instances of the MKP. Operating over a binary string representing a solution to the MKP, Simulated Annealing was used to probabilistically determine whether to invert or swap a variable at a given point of the search. Using Simulated Annealing was shown to outperform deterministically selecting variable changes using a greedy strategy which only accepts changes improving the current solution. Qian and Ding [183] used a similar scheme applied to a larger range of instances, this method was able to outperform the constructive heuristics of Magazine and Oguz [142] and Volgenant and Zoon [208]. Artificial Neural Networks [93] were used to solve a set of MKP instances by Ohlsson et al. [168], showing similar performance to Simulated Annealing on the instances tested. Khuri et al. [123] used a standard Genetic Algorithm to solve a set of 9 instances of the MKP, inspired by real-world problems. Over 100 runs of the Genetic Algorithm, the optimal solution was found at least once in 8 of the 9 instances tested. A Memetic Algorithm operating over a weight-based representation was presented by Cotta and Troya [46]. Cleary and O'Neill [42] used Grammatical Evolution to evolve two types of grammar to solve the MKP, outperforming Khuri et al. [123] in all and Cotta and Troya [46] in 2 of the 9 instances tested. A variety of different representations for Memetic Algorithms solving the MKP were investigated by Raidl and Gottlieb [185]. This study found that the most effective representation within this environment for the MKP was direct encoding as a binary string. Hembecker et al. [100] presented an application of Particle Swarm Optimisation [121] to the MKP. This method failed to find the optimal solution for many of the small instances tested. Two classes of Memetic Algorithms for the MKP were investigated by Özcan and Basaran [170]. A traditional Memetic Algorithm, which applies local search using a hill climber to each individual generated in a Genetic Algorithm and a Multimeme Memetic Algorithm, which manages multiple hill climbing operators in a co-evolutionary framework were tested. Akçay et al. [2] proposed a greedy constructive heuristic based on the 'effective capacity', defined as the number of times an item could be selected without breaking capacity constraints if multiple copies of a single variable were permitted.

Techniques which combine metaheuristics and mathematical programming belong to the emerging research field of *Matheuristics* [143, 186]. Matheuristics have successfully been applied to a variety of problem domains including the MKP, providing some of the best results in the literature. The linear programming (LP) relaxation of the MKP allows the variables x_j from Equation 3.2.3 to take fractional values, rather than being restricted to discrete values of 0 and 1, as shown in Equation 3.3.1:

$$0 \leq x_j \leq 1, \qquad j = 1, ..., n \tag{3.3.1}$$

The LP-relaxed version of the MKP is solvable in polynomial time and can provide useful information about the current problem instance. The LP-relaxed MKP provides good approximations for solutions to the 0-1 integer MKP, indeed some of the best results in the literature are from methods which combine LP-relaxation and heuristics [40, 182, 206]. Chu and Beasley [40] combined a traditional Genetic Algorithm with a repair operator based on the dual variables of the LP-relaxed problem. Raidl [184] used a similar method which used the actual values of the LP-relaxed solution when repairing candidate solutions. Vasquez and Vimont [206] provided the best known results for the largest instances in the ORLib benchmarks of Chu and Beasley [40], extending the work of Vasquez and Hao [205]. Both of these approaches apply Tabu Search to promising areas of the search space derived from LP-relaxed optima, with the improved algorithm fixing additional variables which match the attributes of a 'good' solution. Puchinger et al. [181] explored the *core concept* for the MKP. The core concept was originally presented for the 0-1 knapsack problem by Balas and Zemel [9] and led to a number of successful algorithms for the classic single constraint knapsack problem. The core concept reduces the problem to a subset of decision variables which represent the most difficult items to identify whether or not they are in an optimal solution. This technique relies on the structure of MKP instances and the assumption that a larger proportion of items with high efficiency are included than those with low efficiency. Here efficiency refers to those items which yield a greater profit with respect to the amount of resources consumed in all dimensions. The core concept fixes the variables of high and low efficiency restricting the optimisation to the difficult to place 'medium' efficiency items. Applying CPLEX [110] to the core problems yielded significant improvement in running times compared to the original problem. A Memetic Algorithm and guided Variable Neighbourhood Search are also implemented on the restricted version of the problem, showing better results than when applied to the original problem directly.

Vimont et al. [207] highlight one of the drawbacks of Vasquez and Hao [205] and Vasquez and Vimont [206], namely that it is possible to fix a given variable to a non-optimal value. This work proposed an efficient method to fix certain variables while simultaneously pruning the search tree. The results of this method were able to find a number of unknown optimal solutions and best known solutions for some instances of a standard benchmark library. Wilbaut and Hanafi [213] and Hanafi and Wilbaut [94] both present heuristics based on iterative relaxation of mixed integer programming models, using linear programming to solve sub-problems of a wider MKP problem. Fleszar and Hindi [71] proposed a number of heuristics based on solutions to the LP-relaxed version of the MKP. These heuristics were capable of offering similar performance to the Memetic Algorithm of Chu and Beasley [40] on some larger instances of the MKP. Another method reducing the problem to a restricted subset of key variables was introduced by Angelelli et al. [3]. 'Kernel Search' identifies an initial set of promising items (the *kernel*) and iteratively extends this set, using exact methods to solve the problem associated with a given subset of variables (including the kernel items) to identify items to be added to the kernel. Boussier et al. [18] presented an exact method based on a Branch and Bound strategy able to improve some of the best known results of well-known benchmark instances. Hanafi et al. [95] hybridise a mixed integer programming model with a Variable Neighbourhood Decomposition Search (VNDS) [96] metaheuristic. Mansini and Speranza [145] propose another method similar to those of Puchinger et al. [181] and Angelelli et al. [3], which reduces a given instance of the MKP down to a set of key variables which are expanded during the search. At each step exact methods are applied to solve the sub-problem for a given set of variables. Croce and Grosso [52] presented another core-based/metaheuristic hybrid which is able to outperform some state-of-the-art approaches given limited computational resources.

3.4 Problem instances

A number of benchmarks sets exist for the MKP, each with different properties. SAC-94 is a standard benchmark library of MKP instances taken from a number of papers in the literature often representing real-world examples. These instances are generally small with m ranging from 2 to 30 and n ranging from 10 to 105 with optimal solutions known for all. The properties of these instances are given in Table 3.2(a).

Chu and Beasley [40] noted that the SAC-94 instances are too small to draw meaningful conclusions of an algorithms performance, leading to the proposal of the ORLib instances, generated using a procedure described by Freville and Plateau [73]. This is a widely used benchmark library in the literature and contains 270 instances containing $n \in \{100, 250, 500\}$ variables, $m \in \{5, 10, 30\}$ dimensions and *tightness ratio* $\in \{0.25, 0.50, 0.75\}$. The properties of the ORLib instances are summarised in Table 3.1.

As optimal solutions are not known for all of these instances, performance is often measured using the %-gap distance from the upper bound provided by the solution to the LP-relaxed problem calculated as:

Table 3.1: Summary of the properties of the ORLib set of 270 MKP benchmark instances

Number of Instances	Variables (n)	Dimensions (m)	*tightness ratio*
10	100	5	0.25
10	100	5	0.50
10	100	5	0.75
10	250	5	0.25
10	250	5	0.50
10	250	5	0.75
10	500	5	0.25
10	500	5	0.50
10	500	5	0.75
10	100	10	0.25
10	100	10	0.50
10	100	10	0.75
10	250	10	0.25
10	250	10	0.50
10	250	10	0.75
10	500	10	0.25
10	500	10	0.50
10	500	10	0.75
10	100	30	0.25
10	100	30	0.50
10	100	30	0.75
10	250	30	0.25
10	250	30	0.50
10	250	30	0.75
10	500	30	0.25
10	500	30	0.50
10	500	30	0.75

Table 3.2: The number of objects n and dimensions m in (a) the SAC-94 instances and (b) the Glover and Kochenberger MKP benchmark instances

(a)

Instance	n	m	Instance	n	m
hp1	4	8	pb1	4	27
hp2	4	35	pb2	4	34
weing1-6	2	28	pb4	2	29
weing7-8	2	105	pb5	10	20
sento1-2	30	60	pb6	30	40
weish1-5	5	30	pb7	30	37
weish6-9	5	40	pet2	10	10
weish10-13	5	50	pet3	10	15
weish14-17	5	60	pet4	10	20
weish18-21	5	70	pet5	10	28
weish22-25	5	80	pet6	5	39
weish26-30	5	90	pet7	5	50

(b)

Instance	n	m
GK01	15	100
GK02	25	100
GK03	25	150
GK04	50	150
GK05	25	200
GK06	50	200
GK07	25	500
GK08	50	500
GK09	25	1500
GK10	50	1500
GK11	100	2500

$$100 * \frac{LPopt - SolutionFound}{LPopt} \tag{3.4.1}$$

where $LPopt$ is the fitness value of the LP-relaxed solution to a given problem and $SolutionFound$ is the fitness value of a solution obtained by a particular method.

A third benchmark set was provided by Glover and Kochenberger [88] including much larger instances, with n up to 2500 and m up to 100. Again optimal solutions are not known for all instances so performance is often measured in terms of relative %-gap as described above. Details of the instances contained within the Glover and Kochenberger [88] (GK) set are given in Table 3.2(b). In order to aid fellow researchers in using these libraries, all three benchmark instance sets have been standardised and are available in a unified format at: `http://www.cs.nott.ac.uk/~jqd/mkp/index.html`.

3.4.1 Datasets used by existing approaches for the MKP

A number of existing methods solving the MKP from the literature have been introduced in the previous section. Table 3.3 highlights which of the datasets described in this section are used by each of these authors. Methods marked * were implemented by a second source. Any bracketed tick, i.e. (✓), indicates that a subset of the instances from the given benchmark set were used.

Table 3.3: Summary of the datasets used by existing methods in the literature

Method	Reference	SAC-94	ORLib	GK
Heuristic	Magazine and Oguz [142]*		✓	
Heuristic	Pirkul [180]*		✓	
Simulated Annealing	Drexl [64]	✓		
Heuristic	Volgenant and Zoon [208]*		✓	
Genetic Algorithm	Khuri et al. [123]	(✓)		
Memetic Algorithm	Chu and Beasley [40]	✓	✓	
Memetic Algorithm	Raidl [184]		✓	
Memetic Algorithm	Cotta and Troya [46]	(✓)		
Matheuristic	Vasquez and Hao [205]		(✓)	✓
Matheuristic	Vasquez and Vimont [206]		(✓)	
Memetic Algorithm	Raidl and Gottlieb [185]		✓	✓
Grammatical Evolution	Cleary and O'Neill [42]	(✓)		
Core Concept	Puchinger et al. [181]		(✓)	
Particle Swarm Optimisation	Hembecker et al. [100]	(✓)		
Simulated Annealing	Qian and Ding [183]		✓	
Heuristic	Akçay et al. [2]	(✓)	✓	
Branch and Bound	Vimont et al. [207]		✓	
Memetic Algorithms	Özcan and Basaran [170]	✓	✓	
Matheuristic	Fleszar and Hindi [71]		✓	
Matheuristic	Wilbaut and Hanafi [213]		(✓)	✓
Kernel Search	Angelelli et al. [3]		(✓)	
Branch and Bound	Boussier et al. [18]		(✓)	
Matheuristic	Hanafi et al. [95]		(✓)	✓
Matheuristic	Hanafi and Wilbaut [94]		✓	
Branch and Bound	Mansini and Speranza [145]		(✓)	(✓)
Matheuristic	Croce and Grosso [52]		(✓)	

3.5 Concluding remarks

This chapter briefly introduced the multidimensional knapsack problem (MKP). The MKP is a well-studied optimisation problem which has been of interest to researchers in both the exact and metaheuristic research communities. The MKP provides not only a wide range of benchmark instances with varying properties with which to test our methods, but also a large number of studies in the literature to compare performance with. As introduced in Chapter 2, hyper-heuristics are an emerging class of search methodologies which operate on a search space of heuristics rather than a search space of solutions as with traditional metaheuristic approaches. No previous known work uses selection hyper-heuristics to solve the MKP. Whilst there are a number of methods which use more than one MKP benchmark for comparison, there is no known previous work assessing performance over all three benchmark sets. A core principal of hyper-heuristic research is to 'raise the level of generality' at which search methods operate. The use of three benchmark sets with varying properties gives rise to the opportunity to test generality, where generality is measured in problem diversity, without having to modify the underlying problem domain implementation. A number of low-level heuristics operating on the MKP are used within this thesis. Many are generic operators which can be applied to any binary represented problem however some make use of domain-specific information for the MKP. The nature of the low-level heuristics used varies, with different crossover, mutation and hill climbing heuristics all applied in various chapters. The low-level heuristic sets used vary from chapter to chapter and will be explained in detail in the relevant sections of the thesis. The following chapter presents preliminary work applying generic selection hyper-heuristics to the multidimensional knapsack problem.

CHAPTER 4

A Study of Selection Hyper-heuristics Applied to the Multidimensional Knapsack Problem

4.1 Introduction

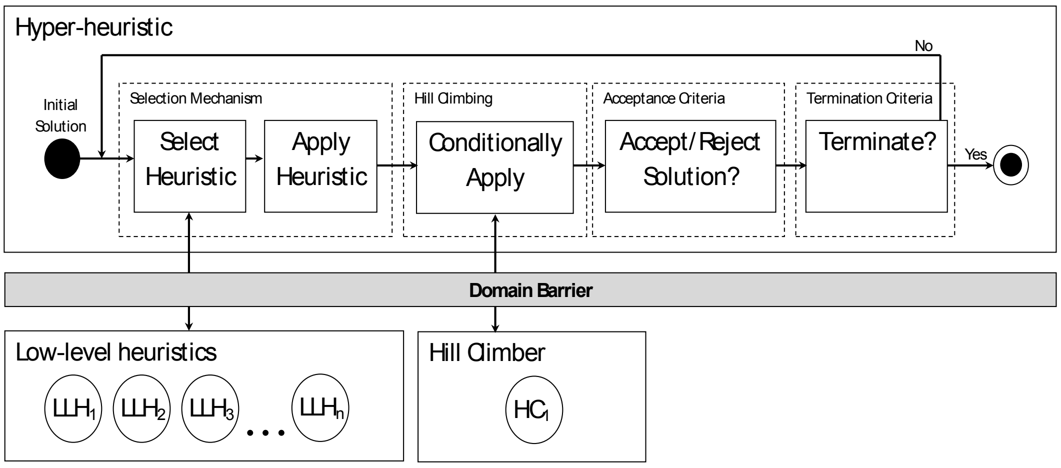
The previous two chapters presented a literature review of hyper-heuristics, a number of real-world optimisation problems and the concept of low-level heuristics which require multiple solutions as input. A number of hyper-heuristic methods solving NP-hard problems were discussed. This chapter presents initial work on solving the multidimensional knapsack problem using selection hyper-heuristics, indicating that hyper-heuristics are a viable solution method for this problem domain. Insights provided by this work will be used in the following chapters to design and refine selection hyper-heuristics operating over multiple problem domains.

The aims of this chapter are twofold. Firstly the suitability of hyper-heuristics to solve the MKP is assessed. Secondly, the relationship between different heuristic selection mechanisms and acceptance criteria pairs is analysed, to provide some insight into the properties of hyper-heuristics which perform well in this domain. A set of generic low-level heuristics which can be applied to any problem domain using a binary representation are implemented. A variety of well-known heuristic selection method and move acceptance criterion combinations are tested.

4.2 Generic selection hyper-heuristics for the MKP

The traditional single-point search hyper-heuristic framework consists of two core components, a heuristic selection method and a move acceptance criterion. Such hyper-heuristics will be labelled *heuristic selection method - move acceptance criterion*, with acronyms used where space is restricted. In this chapter the F_B selection hyper-heuristic framework introduced by Özcan et al. [174] as shown in Figure 4.1 is used.

Figure 4.1: The F_B selection hyper-heuristic framework



In the F_B framework, a high-level search strategy operates on a single solution using a set of mutational and hill climbing low-level heuristics. At each point in the search a low-level heuristic is chosen and applied to an incumbent solution to generate a new solution. In the case where a mutational low-level heuristic has been selected, a pre-defined hill climber is then applied to attempt to improve the new solution. The solution is either accepted and kept as the current solution, or rejected and discarded with the incumbent solution retained as the current solution, according to the move acceptance criterion used. In the original F_B framework, only mutational heuristics were used, in this chapter crossover low-level heuristics are introduced into this framework. Three heuristic selection methods are tested with four acceptance criteria resulting in a total of twelve *heuristic selection method - move acceptance criterion* combinations. The performance of selection hyper-heuristics will be tested on the multidimensional knapsack problem, as introduced in Chapter 3. No existing work applying selection hyper-heuristics to the MKP is known.

4.2.1 Heuristic selection methods and move acceptance criteria used

Three classic heuristic selection methods and four move acceptance criteria are taken from the hyper-heuristic literature to be tested. *Simple Random*, *Choice Function* and *Reinforcement Learning* heuristic selection are combined with *All Moves*, *Only Improving*, *Late Acceptance Strategy* and *Simulated Annealing* move acceptance criteria providing a total of twelve hyper-heuristics. Each of these heuristic selection methods and move acceptance criteria were introduced in Section 2.3.3 and detailed in Table 2.1, Table 2.2 and Table 2.3.

4.2.2 Low-level heuristics

A set of low-level heuristics operating on a binary representation are implemented for each of the hyper-heuristics to select from. The low-level heuristics can be split into three categories: hill climbers, mutation heuristics and crossover heuristics. Hill climbers are *black box* functions which accept a candidate solution as input and guarantee to output a solution which is at least as good. Effectively in this implementation these are local search operators which return a locally optimal solution for a particular neighbourhood structure. In contrast to hill climbers which intensify a search, guiding it closer towards local optima, mutation operators exist to introduce new genetic material and maintain diversity within a search. A mutational heuristic takes a solution as input, performs an operation to perturb the solution and outputs a new solution without quality guarantee. Traditionally, crossover operators are included in population-based approaches as opposed to the single-point search used here. Two solutions are selected from a population and a new solution is generated containing material from both of the original solutions. The general idea is that in a population-based approach only the best quality solutions will survive, containing a mixture of material from good quality solutions.

With the exception of one of the hill climbing heuristics, all of the low-level heuristics implemented are generic operators which can be applied to any problem domain using a binary representation.

Hill climbers

Three hill climbing heuristics are included in the low-level heuristic set. The first two are selected as they have shown promising performance as 'memes' in the work of Özcan and Basaran [170].

Steepest Gradient hill climbing [161] (often referred to as steepest-ascent or steepest-descent hill climbing depending on whether the objective function is being maximised or minimised) systematically checks all neighbours of hamming distance one from a candidate solution. If an improvement is found, the neighbour which provides the greatest increase in the objective value replaces the original solution. The pseudocode for this low-level heuristic in the case of a maximisation problem is shown in Algorithm 5.

Algorithm 5 Steepest Gradient Hill Climbing

```
1: Input $intialSolution$
2: $bestSolution \leftarrow intialSolution$
3: for $i = 1$ to $intialSolution.length$ do
4:     $currentSolution \leftarrow intialSolution$ with bit $i$ inverted
5:     if $fitness(currentSolution) > fitness(bestSolution)$ then
6:         $bestSolution \leftarrow currentSolution$
7:     end if
8: end for
9: return $bestSolution$
```

Davis's Bit hill climbing [54] iterates over a binary string in a random order inverting each bit in turn. If an improvement is found at any stage, the improved solution is accepted as the current solution and the search is resumed at the next bit. If an inversion does not yield an improvement the bit is flipped back before continuing. This is described in Algorithm 6 which requires a random *permutation* of the integers $1, ..., n$, where n is the length of the binary string, as input to define the order in which the bits in the binary string are inverted. Again this algorithm assumes a maximisation problem.

Algorithm 6 Davis's Bit Hill Climbing

```
1: Input $intialSolution, permutation$
2: $bestSolution \leftarrow intialSolution$
3: for $i = 1$ to $intialSolution.length$ do
4:     $currentSolution \leftarrow bestSolution$ with bit $permutation[i]$ inverted
5:     if $fitness(currentSolution) > fitness(bestSolution)$ then
6:         $bestSolution \leftarrow currentSolution$
7:     end if
8: end for
9: return $bestSolution$
```

MKP Hill Climber (MKPHC). A number of papers in the literature (e.g. [40, 180, 142]) make use of an *add* and (or) *drop* phase to either construct, improve or repair solutions to the MKP. These techniques more often than not use a *utility-weight* value to sort the objects in order of their relative efficiency ('bang for buck'). In the case of the classic 0-1 knapsack problem a simple efficiency measure *utility-weight*$_j$ can be defined as:

$$\textit{utility-weight}_j = \frac{p_j}{w_j} \tag{4.2.1}$$

where p_j is the profit provided for including item j in the knapsack and w_j is the resource consumption of this item. Effectively this is the profit-to-weight ratio of a given item. Due to the additional constraints of the multidimensional knapsack problem this must be generalised for the multidimensional case to include the resource consumption in all dimensions:

$$\textit{utility-weight}_j = \frac{p_j}{\sum_{i=1}^{m} w_{ij}} \tag{4.2.2}$$

where w_{ij} is the resource consumption of a given item j in dimension i, in a problem with m dimensions. An issue with this method is that the constraints of a single dimension which dominates all others could distort the weightings. Scaling with respect to the knapsack capacity b_i for each dimension can reduce this effect:

$$\textit{utility-weight}_j = \frac{p_j}{\sum_{i=1}^{m} \frac{w_{ij}}{b_i}} \tag{4.2.3}$$

Using the simple scaled efficiency measure described in Equation 4.2.3 to calculate the relative efficiency of each item *utility-weight*$_j$, a local search operator for the MKP can be implemented. When given an infeasible solution, *drop* items from the knapsack in order of increasing *utility-weight* until a feasible solution is found. When a feasible solution is attained, attempt to *add* items in order of decreasing *utility-weight* until a feasible solution cannot be found by adding another of the unselected items. This operator will be applied as a local search mechanism after each crossover or mutational operator is applied to repair and locally improve solutions, as required by framework F_B described in Section 4.2.

Algorithm 7 MKP Hill Climber (MKPHC)

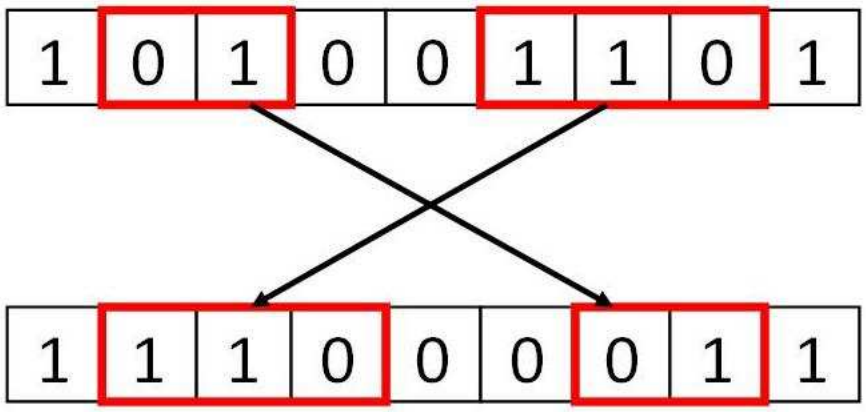
```
1:  Input $solution$, utility-weights
2:  while $solution$ is infeasible do
3:      remove an item from $solution$ in ascending order of utility-weights
4:  end while
5:  for all items not currently added to $solution$ do
6:      add an item to the knapsack in decreasing order of utility-weights
7:      if $solution$ becomes infeasible then
8:          remove the last item added from $solution$
9:      end if
10: end for
11: return $solution$
```

Mutational Operators

Mutational low-level heuristics serve to introduce new 'genetic material' into the search by stochastically modifying a solution in some way. Two simple mutation low-level heuristics taken from the literature are implemented here.

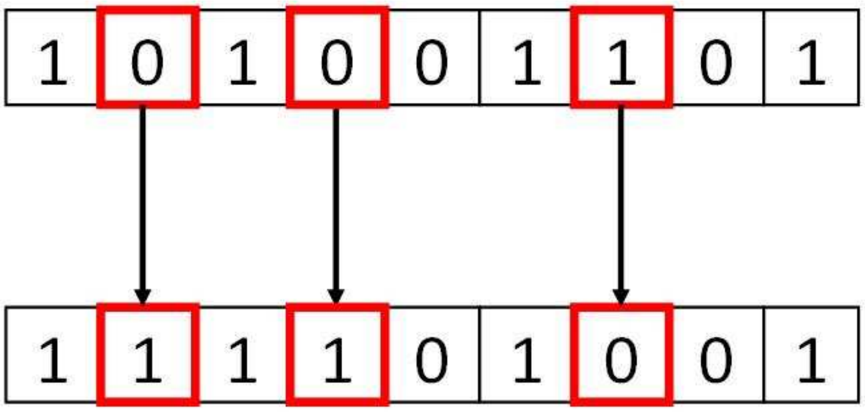
Swap Dimension mutation [173] selects two distinct substrings of a candidate and exchanges their position to generate a new solution, as shown in Figure 4.2.

Figure 4.2: An example of generalised Swap Dimension mutation on a binary string



Paramaterised Mutation inverts a pre-specified number of bits within a string. This is essentially the bit string mutation of Koza [129], however rather than relying on mutational probabilities Parameterised Mutation guarantees the number of bits that are mutated during the operation. An example of Paramaterised Mutation with three bits is given in Figure 4.3.

Figure 4.3: An example of Paramaterised Mutation on a binary string with a mutation rate of 3

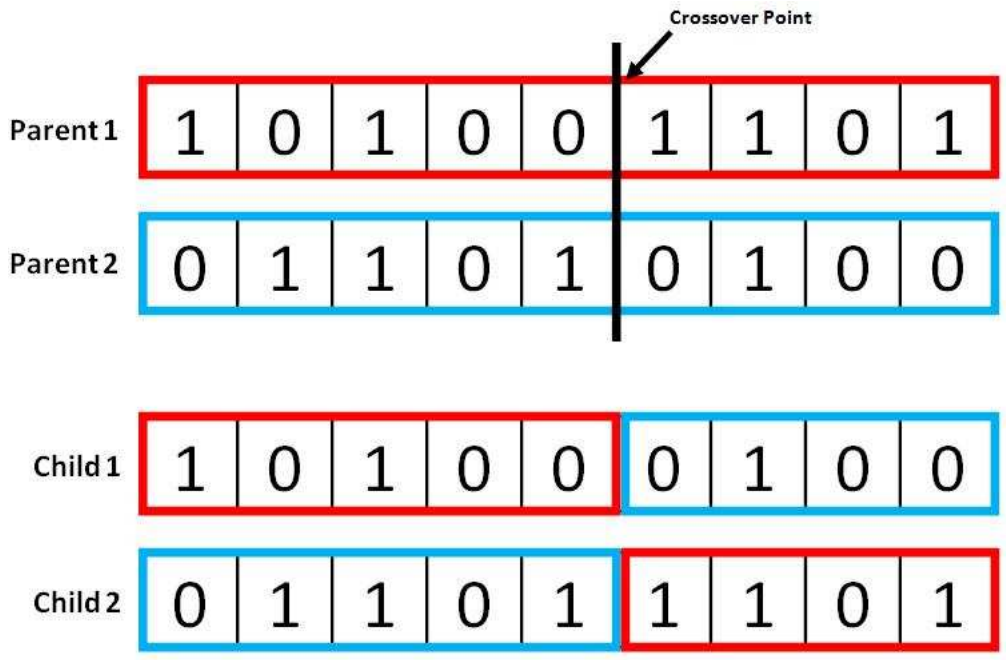


Crossover Operators

In evolutionary algorithms, crossover operators are used to recombine multiple candidate solutions to yield a new solution, which hopefully inherits good genetic material from both solutions. Single-point selection hyper-heuristics operate on a single solution, providing a problem when considering where the second candidate solution will come from. In this framework each operator requires two solutions as input, the first is the current solution in the hyper-heuristic, the second is a randomly generated binary string. This is also known as *Headless Chicken Crossover* [114]. Each operator produces two offspring of which the one with the highest fitness value is kept. There are a large number of crossover operators proposed in the literature for a variety of general and specific purposes. Three of the most simple and well-known are included in this framework. More advanced methods to provide the second solution to be used in operators requiring multiple solutions for input are investigated in Chapter 5.

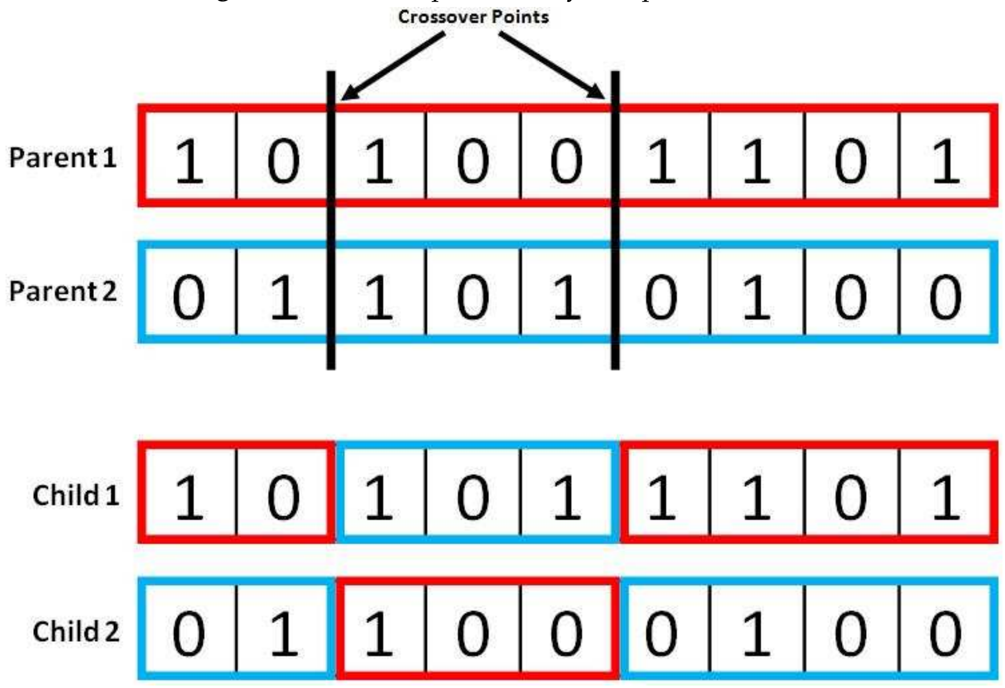
One-point crossover [90] is the most basic binary crossover technique. One-point crossover takes two binary represented solutions as input, selects a single crossover point at random, and exchanges the genetic data that appears on one side of this point between the two solutions. As there is not a natural method to decide which side of the crossover point to swap the genetic material between, it is possible to generate two new strings by swapping either before or after the crossover point. An example with two input (parent) solutions generating two output (child) solutions is shown in Figure 4.4.

Figure 4.4: An example of binary one-point crossover



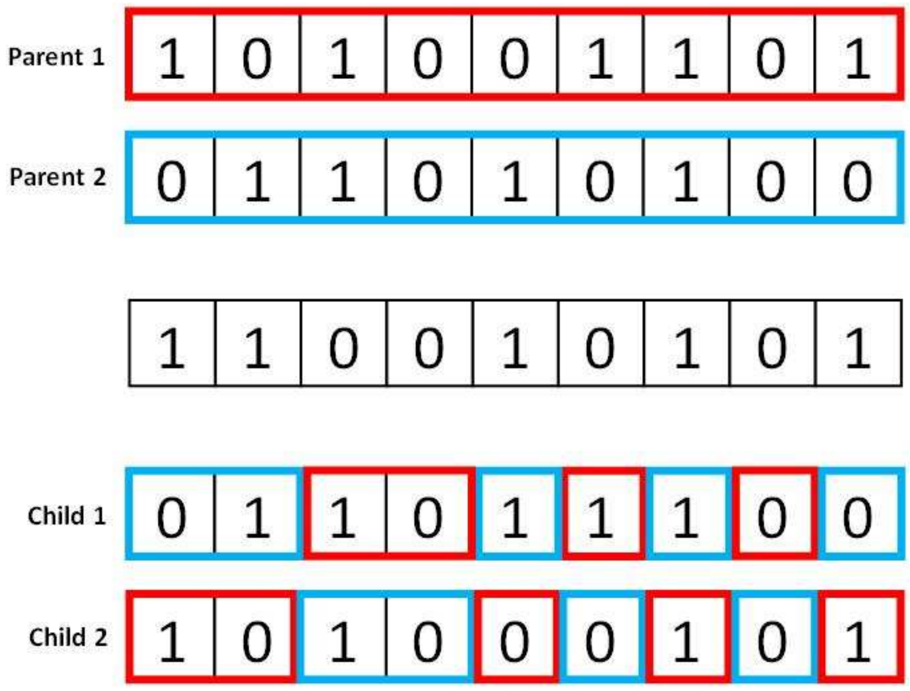
Two-point crossover [90] is similar to one-point crossover, except two crossover sites are chosen and the genetic material that is contained within these two sites is exchanged. This idea can be extended to a general *k*-point crossover, where *k* is the number of crossover sites to be used. Figure 4.5 shows an example of two-point crossover using two parent solutions.

Figure 4.5: An example of binary two-point crossover



Uniform crossover [201] considers each position of two parent input solutions in turn. At each position a bit is selected from one of the solutions with probability p_e, set at 0.5, and copied to the corresponding position in the child solution. Again, it is possible to produce two different children using this method. Uniform crossover is shown in Figure 4.6 with a 0 indicating that the bit should be taken from Parent 1 and a 1 indicating that the bit should be taken from Parent 2.

Figure 4.6: An example of binary uniform crossover



4.2.3 Experimental setup

All experiments are carried out on an Intel Core 2 Duo 3 GHz CPU with 2 GB memory, using the ORLib [40] multidimensional knapsack problem instances as benchmarks. As optimal solutions are not known for all instances, performance is measured using the %-gap measure defined in Section 3.4. Since each set of instances have different ranges of objective function values, ordinal data analysis is also used to compare hyper-heuristics. Given m instances and n hyper-heuristics, an ordinal value O_k is given to each hyper-heuristic for each instance set k, representing the rank of the hyper-heuristic in this set. The total score (Borda count [16]) of a particular hyper-heuristic is the sum of the ranks O_k assigned for each of the m instances tested. The rankings for each instance set are obtained using the average %-gap, calculated using Equation 3.4.1, over the 10 instances in each set. The best performing hyper-heuristic is assigned the lowest rank.

For these experiments, each instance is run until 10^6 fitness evaluations have been performed in order to directly compare results with a number of methods in the literature including the Memetic Algorithms of Chu and Beasley [40], Raidl [184] and Özcan and Basaran [170]. Initial solutions are set as a single random binary string of length n for each $n \in \{100, 250, 500\}$. Twelve hyper-heuristics are tested using three heuristic selection mechanisms and four acceptance criteria.

Heuristic selection method parameters

In the hyper-heuristics using *Reinforcement Learning* heuristic selection, the parameters are derived from Nareyek [162]. The utility values for each low-level heuristic are initially set to 10. In each case the application of a low-level heuristic leads to an improvement in the quality of solution, the utility value for this heuristic is incremented by 1. In the case the application of the low-level heuristic leads to a degradation in solution quality the utility value is decreased by 1. The utility value of an individual low-level heuristic is bound by a maximum value of 30 and a minimum value of 0.

Acceptance criteria parameters

A list length of 500 is used in the *Late Acceptance Strategy*-based hyper-heuristics as suggested by previous approaches [20, 175]. Simulated Annealing calculates the probability p of accepting a solution as defined in Section 2.2.4. The initial value of T is set as the difference between the initial solution and the solution obtained by solving the LP-relaxed version of the problem. During the search process, T is reduced to 0 in a linear fashion proportional to the number of fitness evaluations left.

Low-level heuristic parameters

When the Parameterised Mutation low-level heuristic is chosen mutation is performed at a rate of 10%, i.e. the number of bits inverted is the number of variables in the problem instance n divided by 10. For Swap Dimension the substrings used are fixed to a set number of variables $n/10$, i.e. the substrings are set to 10% of the total length of the solution.

4.3 Results and Discussion

This section presents the results of the selection hyper-heuristics described in this chapter. Section 4.3.1 compares the performance of hyper-heuristics which use the same heuristic selection method. Section 4.3.2 compares the performance of hyper-heuristics which use the same acceptance criterion. In Section 4.3.3, the performance of all twelve hyper-heuristics is presented including a more detailed comparison between the some methods of interest. Finally, the hyper-heuristics are compared to a number of existing methods from the literature in Section 4.3.4.

4.3.1 Performance comparison between hyper-heuristics sharing a common heuristic selection method

The performance of hyper-heuristics which share a common heuristic selection method will be compared in this section. Three heuristic selection methods were used as introduced in Section 4.2.1: *Simple Random*, *Choice Function* and *Reinforcement Learning*. For each of the heuristic selection methods there are four possible move acceptance criteria. In ordinal analysis, each hyper-heuristic is assigned a score O_k between 1 and 4 for each set of 10 instances depending on its relative ranking compared to the other hyper-heuristics. As there are 27 sets of instances the best score possible is 27 and the worst possible score is 108.

***Simple Random* hyper-heuristics**

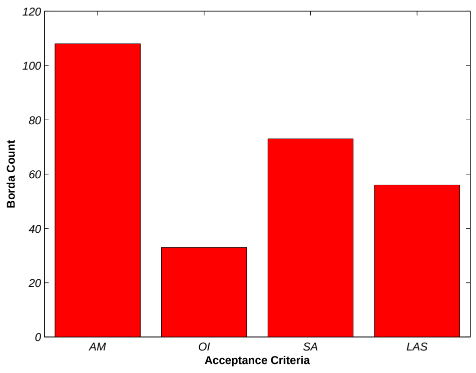
Figure 4.7(a) shows the Borda counts for the four hyper-heuristics which use *Simple Random* heuristic selection. The best hyper-heuristic of this type is *Simple Random - Only Improving* with a score of 33. *Simple Random - All Moves* performs particularly badly compared to the other hyper-heuristics yielding the highest average %-gap in all 27 sets of instances therefore having the maximum Borda count of 108.

***Choice Function* hyper-heuristics**

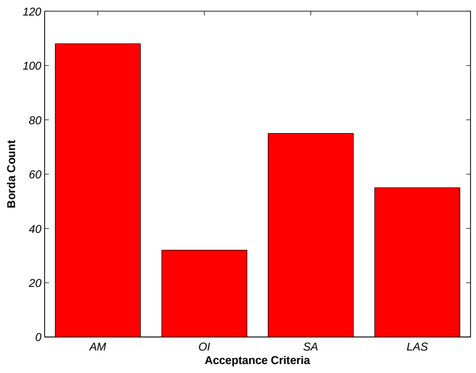
The Borda counts for *Choice Function* selection hyper-heuristics are presented in Figure 4.7(b). *Choice Function - Only Improving* performs very well compared to the other hyper-heuristics with *Choice Function* heuristic selection. Unlike the early work of Cowling et al. [47] where *Choice Function - All Moves* was seen to work well compared to *Choice Function - Only Improving*, *All Moves* acceptance criterion performs very badly in the case. The second best hyper-heuristic of this type is *Choice Function - Late Acceptance Strategy*, with *Choice Function - Simulated Annealing* offering similar performance.

Figure 4.7: Borda counts for hyper-heuristics with a common heuristic selection method

(a) *Simple Random* selection hyper-heuristics



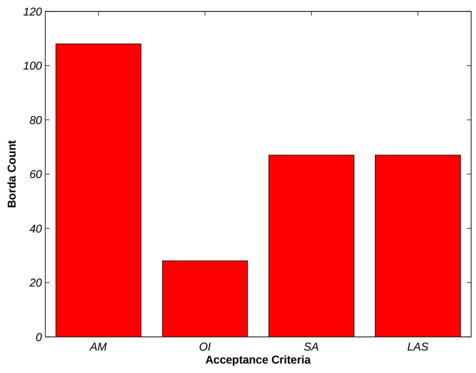
(b) *Choice Function* selection hyper-heuristics



Reinforcement Learning hyper-heuristics

In Figure 4.8 the Borda counts of hyper-heuristics using *Reinforcement Learning* heuristic selection are shown. *Only Improving* is again the best acceptance criteria with a Borda count of 28, obtaining the best average %-gap in 26 of the 27 sets. *Reinforcement Learning - Simulated Annealing* and *Reinforcement Learning - Late Acceptance Strategy* obtain the same Borda counts with scores of 67. As with *Simple Random* and *Choice Function* heuristic selection, *All Moves* acceptance performs very poorly obtaining the worst results in all 27 sets of instances obtaining the maximum Borda count possible of 108.

Figure 4.8: Borda counts for hyper-heuristics using *Reinforcement Learning* as a heuristic selection method



4.3.2 Performance comparison between hyper-heuristics sharing a common acceptance criterion

In the previous section, hyper-heuristics were compared relative to one another based on a shared heuristic selection method. It cannot be ascertained from this information whether there is any difference in the performance between hyper-heuristics which share a common move acceptance criterion. This section compares the performance of hyper-heuristics which use the same move acceptance criterion with different heuristic selection mechanisms. Four move acceptance criteria were used as introduced in Section 4.2.1: *All Moves*, *Only Improving*, *Simulated Annealing* and *Late Acceptance Strategy*. For each of the move acceptance criteria there are three possible heuristic selection methods. In the ordinal analysis each hyper-heuristic is assigned a score O_k between 1 and 3 for each set of 10 instances depending on its relative ranking compared to the other hyper-heuristics. As there are 27 sets of instances the best score possible is 27 and the worst score possible is 81.

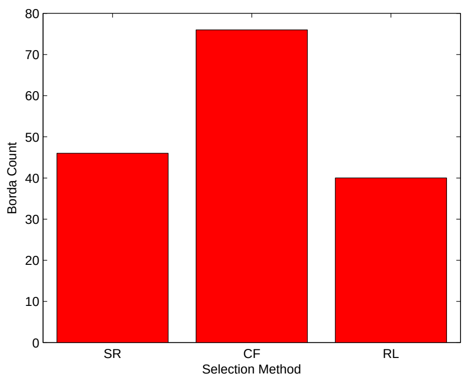
All Moves hyper-heuristics

The *All Moves* hyper-heuristics were relatively the worst performing hyper-heuristics for each of the three heuristic selection methods used in the comparisons of Section 4.3.1. Figure 4.9(a) compares the three *All Moves* hyper-heuristics with differing acceptance criteria. Despite being an intelligent heuristic selection mechanism, the *Choice Function* is outperformed by naive *Simple Random* heuristic selection when using *All Moves* acceptance. Of the three heuristic selection mechanisms, *Reinforcement Learning* performs best when using this move acceptance criterion.

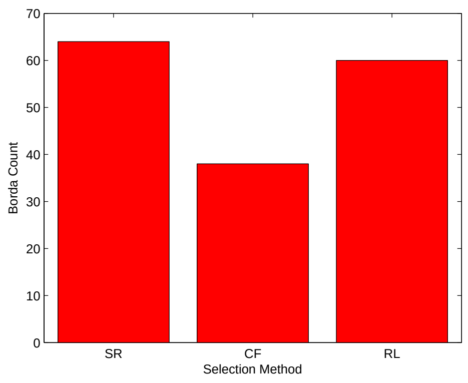
Only Improving hyper-heuristics

Only Improving hyper-heuristics are the best performing hyper-heuristics in the case of *Simple Random*, *Reinforcement Learning* and *Choice Function* heuristic selection methods. Figure 4.9(b) shows the relative performance of the three hyper-heuristics using *Only Improving* move acceptance criterion. *Choice Function - Only Improving* was clearly the best *Choice Function*-based hyper-heuristic in the previous section and again it is the best hyper-heuristic when comparing hyper-heuristics with the same move acceptance criterion, scoring 38 points in total. Despite not being the best *Simple Random* hyper-heuristic, *Simple Random - Only Improving* still offers similar performance to *Reinforcement Learning - Only Improving*, scoring 64 and 60 points in the Borda counts respectively. Again, it is observed that a naive heuristic selection method can perform

Figure 4.9: Borda counts for hyper-heuristics with a common move acceptance criterion



(a) *All Moves* selection hyper-heuristics



(b) *Only Improving* selection hyper-heuristics

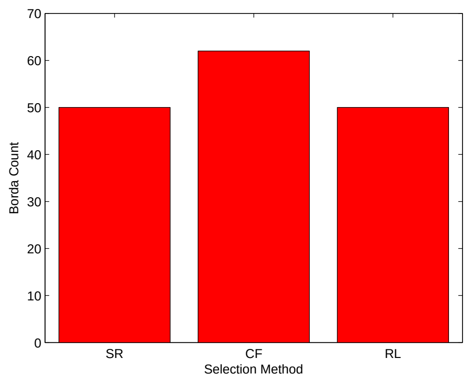
comparably to a more 'intelligent' mechanism when coupled with a certain acceptance criterion. In terms of the number of pure best results for each of the 27 sets, *Choice Function - Only Improving* offers the best performance in 19 sets, *Reinforcement Learning - Only Improving* in 6 sets and *Simple Random - Only Improving* gives the best performance in 2 sets.

Simulated Annealing hyper-heuristics

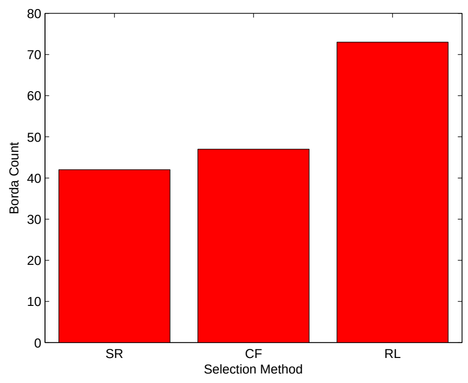
In relative terms, the *Simulated Annealing*-based hyper-heuristics were the second worst performing hyper-heuristics in Section 4.3.1. Figure 4.10(a) compares the performance of the three *Simulated Annealing* hyper-heuristics based on Borda counts, again a lower score is better, with a minimum possible score of 27. *Simple Random - Simulated Annealing* and *Reinforcement Learning - Simulated Annealing* score exactly the same with 50 points. Closer inspection of these results reveals that not only do these two hyper-heuristics score identical when using Borda counts, they also are the best performing hyper-heuristic in the same number of datasets, both providing the best results for 11 sets of instances. *Choice Function - Simulated Annealing* performs slightly worse with a Borda count of 62. With *Simulated Annealing* there is less variation between the performance of hyper-heuristics using different heuristic selection methods than with other acceptance criteria, suggesting that the heuristic selection method used has less of an effect when combined with *Simulated Annealing* than other acceptance criteria.

Figure 4.10: Borda counts for hyper-heuristics with a common move acceptance criterion (ii)

(a) *Simulated Annealing* selection hyper-heuristics



(b) *Late Acceptance Strategy* selection hyper-heuristics



Late Acceptance Strategy hyper-heuristics

Figure 4.10(b) shows the relative performance of the three *Late Acceptance Strategy*-based hyper-heuristics. *Simple Random* heuristic selection combined with *Late Acceptance Strategy* outperforms both *Choice Function* and *Reinforcement Learning* heuristic selection. The *Simple Random - Late Acceptance Strategy* hyper-heuristic has a Borda count of 42 with the *Choice Function* and *Reinforcement Learning*-based hyper-heuristics scoring 47 and 73 respectively. *Simple Random - Late Acceptance Strategy* has been shown to work well previously by Misir et al. [153] and Jackson et al. [112].

4.3.3 Performance comparison of all hyper-heuristics

The %-gap was introduced as a performance measure of methods solving the MKP in Section 3.4. The results for each of the twelve hyper-heuristics in terms of average %-gap are presented in Table 4.1. Each result reported is the average of the %-gaps of all 27 sets of 10 instances contained in the OR-Lib benchmark set.

When considering the raw average %-gap, the best performing hyper-heuristic is *Choice Function - Only Improving*. This is closely followed by four more hyper-heuristics with very similar average %-gaps. Following a Shapiro-Wilk test [193] providing confirmation that the set of results for each hyper-heuristic is not normally distributed, a Mann-Whitney U test [144] within a 95% confidence interval is performed. The results indicate that there is no statistically significant difference between the results of *Choice Function - Only Improving* and the second and third placed hyper-heuristics, *Re-*

Table 4.1: Overall performance of each hyper-heuristic over OR-Lib benchmarks in terms of average %-gap

Hyper-heuristic	%-gap
Choice Function - Only Improving	2.16
Reinforcement Learning - Only Improving	2.35
Simple Random - Only Improving	2.39
Choice Function - Late Acceptance Strategy	3.31
Simple Random - Late Acceptance Strategy	3.34
Simple Random - Simulated Annealing	3.62
Reinforcement Learning - Simulated Annealing	3.63
Reinforcement Learning - Late Acceptance Strategy	3.67
Choice Function - Simulated Annealing	3.68
Reinforcement Learning - All Moves	4.97
Simple Random - All Moves	5.07
Choice Function - All Moves	5.39

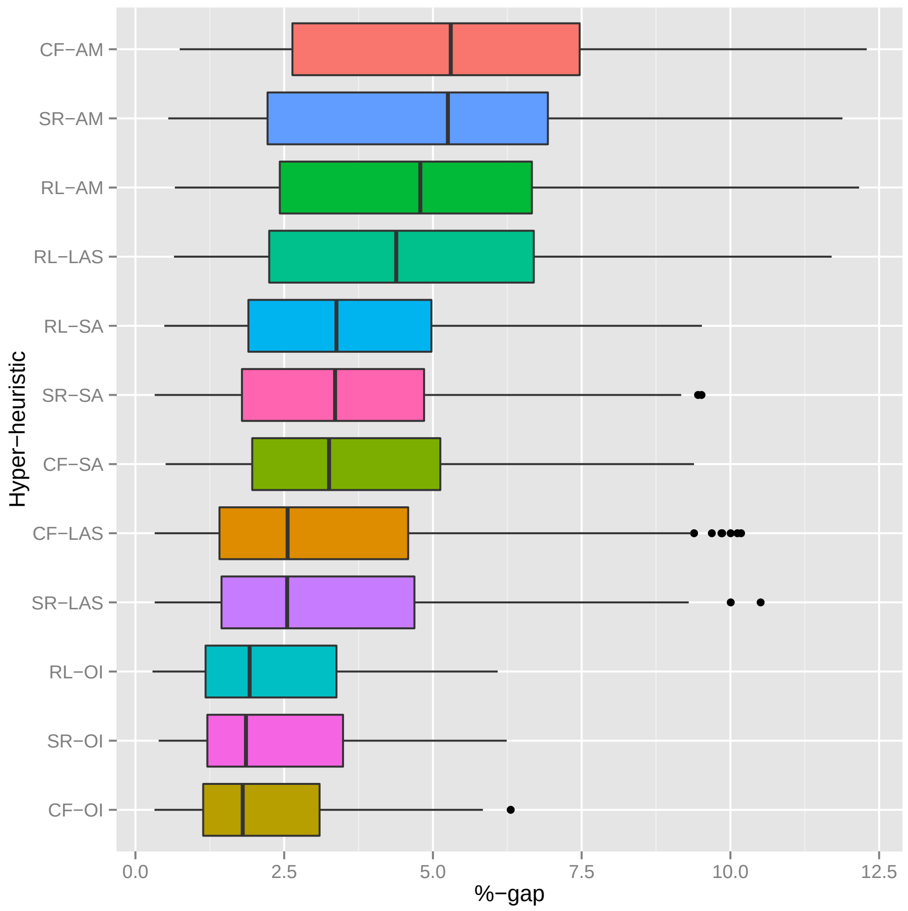
inforcement Learning - Only Improving and *Simple Random - Only Improving* over the 270 instances tested.

In general, those hyper-heuristics which use *Only Improving* move acceptance offer the best performance in terms of average %-gap. *Late Acceptance Strategy* and *Simulated Annealing* hyper-heuristics offer relatively average performance whilst the hyper-heuristics using *All Moves* move acceptance criterion perform particularly badly. The performance of *All Moves* acceptance is perhaps unsurprising as this mechanism makes no attempt to guide the search, potentially leading to too much diversity in the search process.

Despite being intelligent heuristic selection mechanisms designed with the intention of 'learning' which heuristic to select at a given point in the search, *Choice Function* and *Reinforcement Learning* hyper-heuristics are often outperformed by *Simple Random* heuristic selection. In fact *Simple Random* hyper-heuristics account for 2 of the 4 best methods found. This is in line with the comments of Misir et al. [151] who observed that 'intelligent' mechanisms for heuristic selection may not necessarily perform as well as simple ones under a given set of experimental circumstances.

Figure 4.11 provides a box and whisker comparison of the twelve hyper-heuristics in terms of the individual %-gaps obtained for all 270 instances of ORLib. This plot highlights the consistency in performance by those methods using *Only Improving* move

Figure 4.11: Box and whisker comparison of twelve hyper-heuristics over all 270 OR-Lib instances in terms of %-gap



acceptance criterion rather than, for example, *All Moves* acceptance by illustrating the spread of %-gap values obtained. All methods which share a move acceptance criterion perform very similarly with the exception of *Reinforcement Learning - Late Acceptance Strategy* which is outperformed by the other two *Late Acceptance Strategy*-based hyper-heuristics. When sorting the hyper-heuristics by median %-gap value in this way, the leading methods are in a slightly different order in terms of performance. *Simple Random - Only Improving* is now above *Reinforcement Learning - Only Improving* and *Simple Random - Late Acceptance Strategy* is now above *Choice Function - Late Acceptance Strategy*. In addition, *Choice Function - Simulated Annealing* moves from ninth to sixth place. A Mann-Whitney U test within a 95% confidence interval indicates that there is no statistically significant difference between any of the methods which exchange places when using median %-gap rather than average %-gap.

Detailed comparison of hyper-heuristic performance

The behaviour of two of the best hyper-heuristics in the previous section, *Choice Function - Only Improving* and *Reinforcement Learning - Only Improving*, will be analysed in more detail in this section. As discussed in Section 4.3.3, there is no statistically significant difference between these two hyper-heuristics in terms of average %-gap. Figure 4.12 provides a plot of both the average current 'working' and average best found solution, with respect to the number of fitness evaluations, of 10 runs of each hyper-heuristic on ORLib instance ORLib30x500_0.75-01 using a fixed starting point. This instance is taken from the largest set of instances in ORLib and is the first instance from the set with 30 dimensions, 500 variables and a *tightness ratio* of 0.75. The optimal solution to this problem is still unknown, 15 years after it was first proposed by Chu and Beasley [40]. The 'working' solution is defined as the fitness value of the last solution generated before it is considered by the move acceptance criterion.

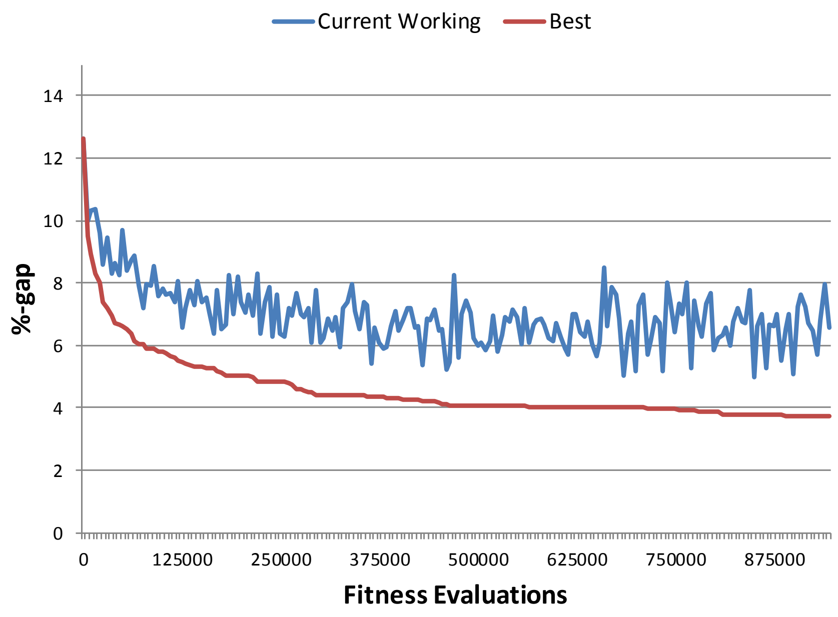
From this figure, it is noted that the performance of the two hyper-heuristics again appears very similar. Following an initial sharp drop in %-gap value, the fitness of the best solution found so far is gradually improved throughout the search. Although the improvements become less drastic in the later phases of the search, there are still clearly best-of-run solutions being found towards the end of the run. This suggests that the search may not have converged, and that more fitness evaluations are required for these hyper-heuristics to reach the best solutions they are capable of finding on a consistent basis. *Choice Function - Only Improving* seems to be searching a slightly greater range of values, although both hyper-heuristics are operating over a very similar set of %-gap values ranging roughly between 6% and 8%.

As stated previously, the move acceptance criterion used appears to have a greater effect on performance than heuristic selection method, when solving the MKP benchmarks with the hyper-heuristics tested. In light of this it may be more interesting to compare the performance of the best hyper-heuristic, *Choice Function - Only Improving*, with other hyper-heuristics sharing this heuristic selection method. Figure 4.13 shows the plots for the same instance as before, using *Choice Function - Simulated Annealing* and *Choice Function - Late Acceptance Strategy*.

In these two figures the acceptance mechanism used is evident in the shape of the plots generated. Ignoring the initial dip in %-gap value which is necessary to move the search from the starting area of the search area to one which is more promising, the acceptance mechanism used becomes visible. *Simulated Annealing* will accept non-improving moves with a decreasing probability over the course of a search. In the early

Figure 4.12: Average working and best %-gap values with respect to number of evaluations for MKP instance ORLib30x500_0.75-01

(a) *Choice Function - Only Improving*



(b) *Reinforcement Learning - Only Improving*

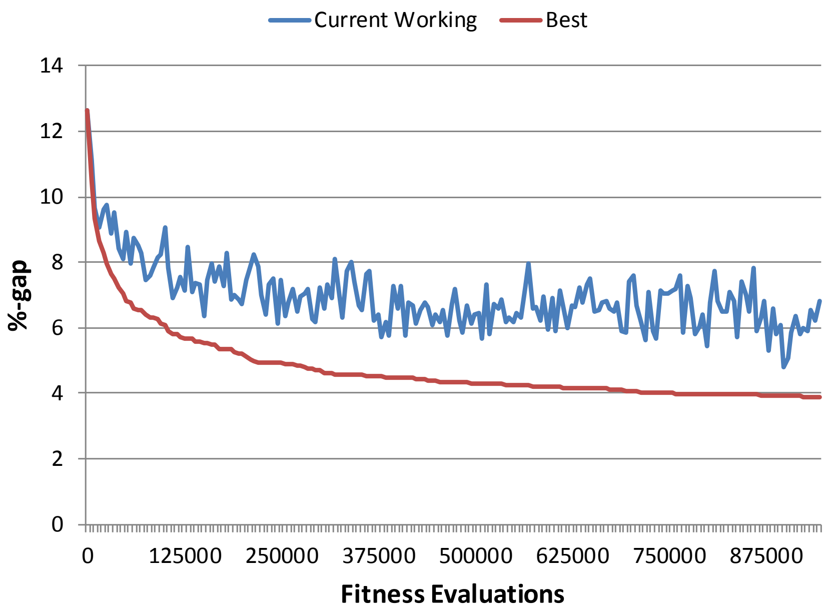
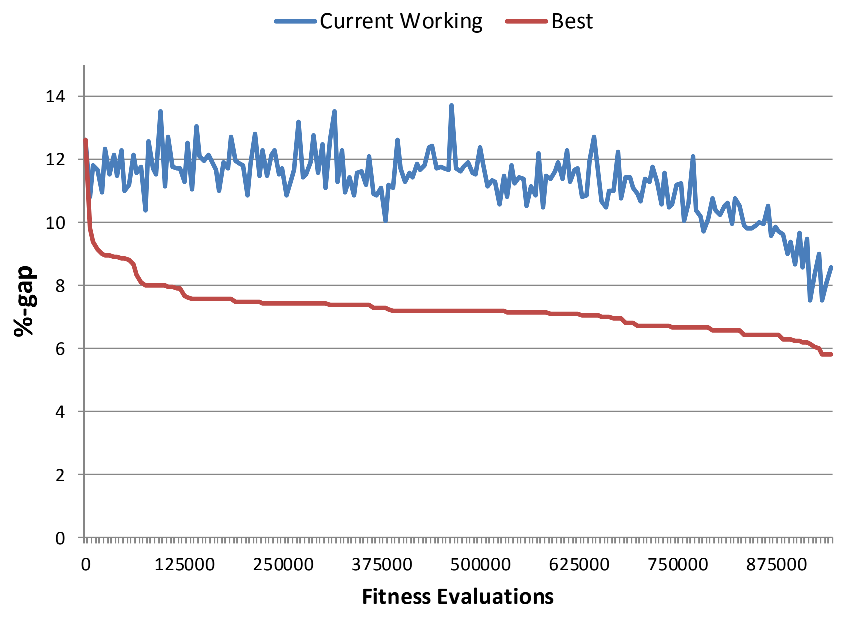
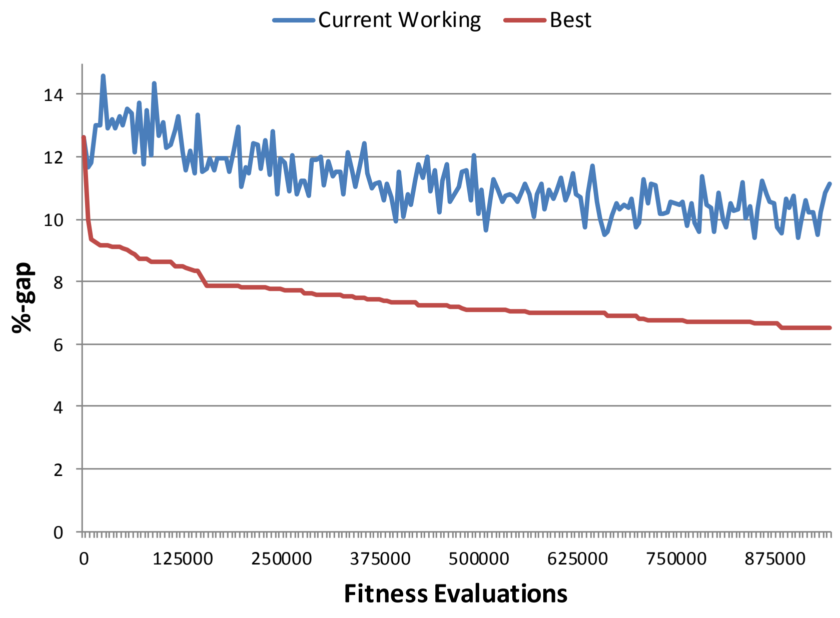


Figure 4.13: Average working and best %-gap values with respect to number of evaluations for MKP instance ORLib30x500_0.75-01 (ii)

(a) *Choice Function - Simulated Annealing*



(b) *Choice Function - Late Acceptance Strategy*



phases non-improving moves are more likely, before the search is intensified in later phases behaving more like *Only Improving* acceptance. In Figure 4.13(a) the gradual intensification can be seen, with an increased number of best-of-run solutions being found towards the end of the run. Figure 4.13(b) also shows evidence of the acceptance mechanism used. *Late Acceptance Strategy* promotes a general trend of improvement throughout a search, delaying the value of the criteria used to decide whether to accept a solution. In this figure a steady improvement in terms of the working solution and best solution can be seen. In both cases, it seems that the search has not yet reached a point of stagnation within 10^6 fitness evaluations. It could be the case that an increase in performance could be observed by allowing the hyper-heuristics more time to run. This could potentially undermine one of the fundamental principles of hyper-heuristics, to obtain solutions which are 'good enough, soon enough, cheap enough' [21], so methods to guide hyper-heuristics to promising search regions could be a more appropriate way to improve performance.

4.3.4 Comparison with existing approaches to the MKP from the literature

In the previous sections, the best performing selection hyper-heuristic observed was *Choice Function - Only Improving*. In this section the performance of this hyper-heuristic will be compared to a number of existing approaches taken from the literature. As 10^6 fitness evaluations are allowed for each run, it is possible to perform a direct comparison with the methods from the literature which use the same termination criterion. Table 4.2 provides the performance of a number of techniques taken from the literature as a base for comparison.

Although this table shows that *Choice Function - Only Improving* is capable of outperforming some existing methods in the literature, it is not as good as some of the state-of-the-art metaheuristics. *Choice Function - Only Improving* offers better average solution quality than the classic methods of Magazine and Oguz [142] and Volgenant and Zoon [208], the greedy algorithm provided by Akçay et al. [2], CPLEX 4.0 presented by Chu and Beasley [40] and the Simulated Annealing heuristic of Qian and Ding [183]. The best performing method of Chu and Beasley [40] uses the solutions to the LP-relaxed version of the MKP to initialise and locally improve solutions during the evolutionary process. The extra computational effort to calculate the solution to the LP-relaxed variant of the problem is not taken into consideration when comparing based on number of fitness evaluations. Additionally some ambiguous language is used to describe the number of states visited. Chu and Beasley [40] search 10^6 'non-duplicate' states however it is unclear whether or not the solutions at each step of the local improvement phase are counted. *Choice Function - Only Improving* searches 10^6 individuals including duplicate states and the individual steps of the hill climbing low-level heuristics.

Table 4.2: Performance of other approaches over OR-Lib benchmarks

Type	Reference	%-gap
MA	Raidl [184]	0.53
MA	Chu and Beasley [40]	0.54
MA	Özcan and Basaran [170]	0.92
Permutation GA	Hinterding [101] (Raidl [184])	1.30
Heuristic	Pirkul [180]	1.37
Heuristic	Freville and Plateau [73]	1.91
Hyper-heuristic	**Choice Function - Only Improving**	**2.16**
Heuristic	Qian and Ding [183]	2.28
MIP	Chu and Beasley [40]	3.14
Heuristic	Akçay et al. [2]	3.46
Heuristic	Volgenant and Zoon [208]	6.98
Heuristic	Magazine and Oguz [142]	7.69

4.3.5 Remarks and potential future improvements

All three heuristic selection mechanisms work well with *Only Improving* move acceptance criterion. Little can be done to improve the results using this move acceptance criterion through parameter tuning as it has no parameters. Attempts to improve the hyper-heuristics using *Only Improving* acceptance could be made by reducing the number of underlying low-level heuristics, removing poor performing heuristics to reduce the heuristic search space. Although over the whole dataset *Late Acceptance Strategy* does not appear to be particularly successful, it is in fact at least as good as *Only Improving* for all three heuristic selection mechanisms in the instances which contain only 100 objects, i.e. the smallest instances. It may be that different list lengths are suitable for different length instances, further experimentation into using different list lengths could support this. *Simulated Annealing* also struggles on these instances, however it has been improved previously by Bai et al. [8] by adding a re-heating scheme to accept a greater range of solutions at points where the search may have stagnated. There are many such modifications to the parameters of *Late Acceptance Strategy* and *Simulated Annealing* which could potentially improve the performance of hyper-heuristics using these acceptance criteria. In general, accepting *All Moves* is a poor move acceptance strategy when using each of the three given heuristic selection mechanisms on this problem domain.

The parameters for the heuristic selection methods and move acceptance criteria used in this chapter were decided either experimentally, or based on the experience of existing research in the literature. Unfortunately this gives no guarantee that the values chosen are the best values for the current problem instance under consideration. It is likely that the best parameter settings are not only dependent on the *heuristic selection method - move acceptance criterion* combination, but also the problem domain used, the low-level heuristics available and the individual instance currently being solved. A potential future research question would be to analyse the effect of parameter tuning within this framework. Many methods exist in the literature to tune such parameters automatically, a comparison of some of the most popular of these methods is given by Montero et al. [158].

4.4 Concluding Remarks

Hyper-heuristics are intelligent methods which can be used to select appropriate low-level heuristics at a given point of a search space. A number of hyper-heuristics operating on a single-point search framework have been applied to a standard optimisation problem, the multidimensional knapsack problem (MKP). The following chapter will build on the work presented in this chapter by introducing problem-specific knowledge into the low-level heuristic set. Selection hyper-heuristics will be hybridised with linear programming through the use of solutions to the relaxed MKP in two ways. The LP-relaxed solutions will be used within a new initialisation method for the MKP and to guide the hill climbing repair operator within the framework. A number of frameworks in which to manage crossover operators within selection hyper-heuristics will be explored.

CHAPTER 5

A Case Study of Controlling Crossover in a Selection Hyper-heuristic Framework with MKP

5.1 Introduction

In the previous chapter, a number of preliminary experiments were performed using selection hyper-heuristics over a set of generic binary heuristics to solve the MKP. A mixture of mutational, crossover and hill climbing operators were included. This chapter will use an extended version of the framework developed in Chapter 4 to analyse crossover control in selection hyper-heuristics.

In this chapter, the management of crossover in single-point search hyper-heuristics is investigated. Two frameworks are proposed to control crossover in single-point hyper-heuristics and tested on the multidimensional 0-1 knapsack problem (MKP). A number of hyper-heuristics are applied to three benchmark libraries of varying properties from the literature.

Previous work [40, 181, 182, 206] has shown that hybridising linear programming with metaheuristic techniques can yield very good solutions to MKP benchmarks. No known previous work attempts to combine knowledge gained from LP-relaxation within a hyper-heuristic framework. Much of the previous work mentioned includes intelligent mechanisms to initialise solutions using greedy methods based on LP-relaxed solutions. Here a greedy initialisation technique is used to generate a list of potential solutions to use as inputs for crossover low-level heuristics. A new initialisation method which allows both feasible and infeasible solutions is proposed and compared to two initialisation methods from the literature which produce only feasible solutions.

CPLEX [110] is a general-purpose mixed-integer programming (MIP) package used to solve linear optimisation problems. As CPLEX has evolved, a large number of heuristics are now used within the solver to aid the optimisation process. Chu and Beasley [40] provided results using CPLEX 4.0 over a set of MKP benchmark instances. CPLEX 12.5 is a vastly improved solver over CPLEX 4.0, here the results for CPLEX 12.5 are provided for the three benchmark sets described in Section 3.4. These results are provided as a benchmark for comparison for future researchers in this area.

The hyper-heuristics implemented operate over a generic binary representation, so it is possible to apply the hyper-heuristics defined over any problem domain using this representation. As such, the best hyper-heuristic found is applied to a second problem domain; the Boolean satisfiability problem (SAT). This hyper-heuristic is compared to two existing hyper-heuristic methods from the literature operating on this problem domain.

5.2 Controlling crossover in selection hyper-heuristics

Hyper-heuristics have previously been shown to be successful in a wide range of NP-hard optimisation problems as discussed in Chapter 2. As interest in hyper-heuristic research increases, the use of general purpose hyper-heuristic frameworks such as HyFlex [27, 166] and Hyperion [200] is growing. Such frameworks often contain crossover low-level heuristics however the management of these operators is often overlooked, despite being a cornerstone of evolutionary computation for a considerable period of time. As the definition of hyper-heuristics creates a distinct separation between the high-level search strategy and the problem domain level, the question of which level should be responsible for managing the solutions required as input for crossover operators is an open research issue. This also raises a more general question regarding the role memory has to play, specifically whether or not the low-level heuristics in a hyper-heuristic system should be stateful or stateless.

Crossover is often used in population-based metaheuristics as a mechanism to recombine multiple solutions. Two candidate solutions are selected from a population and a new solution is generated containing material from both solutions. The general idea is that in a population-based approach only the best quality solutions will be kept in the population, containing a mixture of material from previous solutions. Little research effort has gone into designing methods to select the solutions as input (more specifically the second input) for crossover low-level heuristics in a single-point selection hyper-heuristic framework.

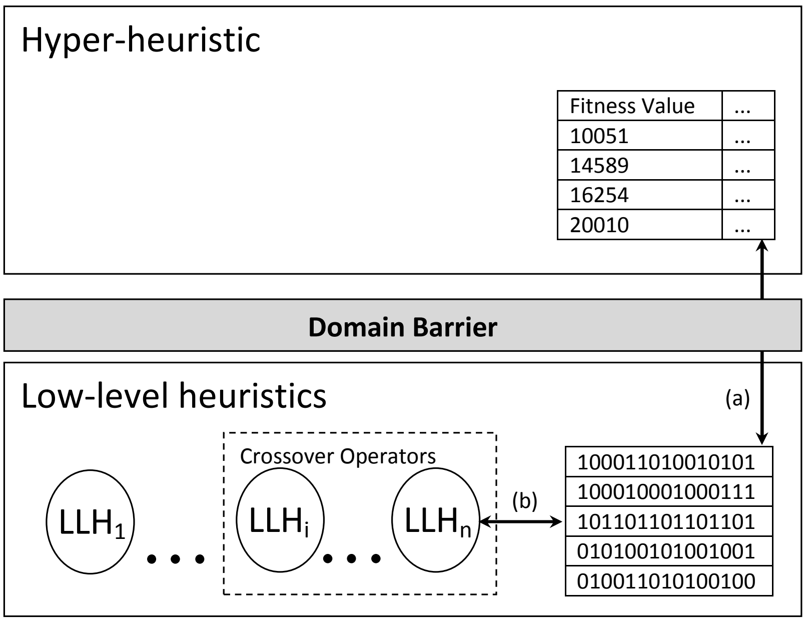
In Chapter 4, the first input used in a binary crossover operator is the current solution and the second input is generated randomly. This does not fit in with the original ethos of crossover which is to preserve and exploit the good characteristics of suitably fit solutions. This crossover operator therefore acts as a mutation operator, by introducing new material to the current solution from a solution which is not necessarily of good quality. In order to make crossover in single-point search behave in a way similar to population-based approaches, a memory of suitably fit individuals to recombine the current solution with can be included. Such approaches which provide the second solution needed for an operator are not just limited to crossover. Any operator which requires more than one input solution where there is not a 'natural' choice for the second input can be managed in this way.

As it is not obvious where the second candidate solution for crossover should be managed in a single-point selection hyper-heuristic, two frameworks are proposed. In each case, a list of potential solutions for crossover is maintained. The general shared framework is shown in Figure 5.1, with a set of crossover low-level heuristics $LLH_i, ..., LLH_n$ operating on set of candidate solutions represented as binary strings. This list does not necessarily need to be restricted to crossover, it could be used by any operator which requires multiple solutions as a means of managing input.

The first framework maintains a list at the hyper-heuristic level. Although the candidate solutions exist below the domain barrier, the hyper-heuristic decides which solution to use for crossover based on feedback given during the search. This raises a number of questions including: which information should be passed back to the hyper-heuristic, how should this list be maintained and how long should this list be? This approach could be considered as adding stateful behaviour to the low-level heuristics. As the low-level heuristics are given a memory, and therefore some notion of state, the behaviour of crossover low-level heuristics will depend on previous low-level heuristic invocations. The interaction between the hyper-heuristic and the solutions is depicted by arrow (a) in Figure 5.1.

The second framework allows the low-level heuristics to manage the list of second solutions for crossover directly. Again this poses similar questions regarding the size of such a list and how it should be initialised and maintained. This framework can be considered as stateless, as effectively the input arguments are hard-coded in a pre-

Figure 5.1: A general framework for controlling crossover with hyper-heuristic control shown by arrow (a) and low-level control shown by arrow (b)



processing phase, with performance not dependent on previous invocations of low-level heuristics. This framework is also shown in Figure 5.1, with the interaction between the low-level heuristics and the solutions depicted by arrow (b). Figure 5.1 should be viewed as an extension to the general hyper-heuristic framework introduced in Section 2.3.3.

5.2.1 Controlling crossover at the hyper-heuristic level

Candidate solutions for use as inputs for crossover operators can be controlled at the hyper-heuristic level. In *Memory Crossover*, a list (memory) is kept of all solutions which were the current best-of-run when found, from which second inputs can be chosen for crossover low-level heuristics. Initially this list is populated randomly. Each time a new best-of-run solution is found it replaces the worst existing solution in the list. A method of choosing a solution to use from this memory is needed. Any evolutionary algorithm selection method can be used for this, here *tournament selection* is preferred. In tournament selection, a subset of solutions of a given *tournament size* is chosen from a list. These solutions are paired up and the strongest (i.e. highest quality) solution in a pair is kept and the other discarded. The pairing process continues until a single solution is left. Tournament selection is a stochastic selection method which favours stronger solutions however lower quality solutions do still have some chance of selection. This method of crossover control is inspired by steady state Genetic Algorithms which select and update a population in much the same way.

As a base for comparison in order to see what benefit is gained by controlling crossover in this way, other methods of choosing the second input solution are also tested. *Random Crossover*, also known as Headless Chicken Crossover [114] and used in Chapter 4, takes the two inputs to be the current solution in the hyper-heuristic and a randomly generated binary string. In this case, the list of potential solutions for crossover is a single random solution. In addition, each hyper-heuristic is also tested with crossover low-level heuristics omitted completely.

5.2.2 Controlling crossover at the domain-specific level

It is also possible to maintain a memory such as the one described at the hyper-heuristic level at the domain-specific level. Here, problem-specific heuristics are used to populate a static list of candidate solutions generated based on problem domain-specific knowledge. One of these solutions is then used as the second input during a crossover operation. The list is static since it is expected that the solutions in the list will contain the 'building blocks' of high quality solutions. This is implemented as a queue of solutions whereby each time a solution is required for crossover the solution at the head of the queue is taken. This solution will be used in the crossover operation before being placed at the tail of the queue. Some procedure must be defined to initialise this list.

A number of methods exist in the literature to initialise solutions for the MKP. Gottlieb [92] compared a number of initialisation methods for evolutionary algorithms solving the MKP. This study focussed on initialisation methods which exploit the property that many good, or near-optimal solutions for the MKP, lie close to the the border between feasible and infeasible solutions. One of the initialisation routines C*, is a variation of the method of Chu and Beasley [40] where starting with an empty solution the algorithm attempts to add each item in a random order. If selecting this item results in a feasible solution, this change is kept and the next item for selection is considered. If an infeasible solution is obtained, this change is reversed and the next item for selection is considered. Another - R*, based on a method originally proposed by Raidl [184], uses the solutions to the LP-relaxed version of each problem to construct each candidate solution. The solution begins with no items selected and is traversed in a random order. For each item in the solution, an attempt to add this item to the solution is made probabilistically based on its value in the LP-relaxed solution. For example if an item has a value of 1 or 0 in the LP-relaxed solution, the initialisation routine will always or never attempt to add this item to the solution respectively. For an item with a fractional value within the LP-relaxed solution, it will be selected with probability equal to this value. Again this change is kept if a feasible solution is obtained and reversed otherwise, before moving on to the next item. A second pass across the items is then made and an attempt to add each item which was not selected in the first iteration is made. This second phase attempts to take the solution closer to the boundary of feasibility and also gives items with value 0 in the LP-relaxed solution a chance of being selected. A potential drawback of both of these approaches is that as only feasible solutions can be generated, there are a large number of infeasible solutions close to the optimal solutions which are not considered.

A new initialisation method allowing infeasible solutions for the MKP

Here a new initialisation method allowing infeasible solutions, *jqdInit*, is proposed. This method is shown in Algorithm 8. Given a solution $S \in \{0,1\}^n$ starting with no items selected, each item j is considered sequentially from left to right. An item is included in the solution with probability equal to the item's value in the LP-relaxed solution, irrespective of whether a feasible solution is obtained or not. Pseudo-random numbers R_j ($0 \leq R_j < 1$) are used in this step. In terms of time complexity, all three initialisation methods must visit every variable once and so are asymptotically equivalent running in $O(n)$ time, where n is the number of variables. Using this initialisation method it is possible to generate both feasible and infeasible solutions to the MKP. In the case of the MKP, optimal solutions lie close to the boundary of feasibility. This means that it is possible for an infeasible solution to be very close to the optimal solution in terms of hamming distance, despite the fact that this is not reflected in the fitness function used. It is therefore important that infeasible solutions are considered, as otherwise a large number of solutions which contain good 'building blocks' are potentially ignored.

Algorithm 8 Algorithm to generate MKP solutions allowing infeasibility (*jqdInit*)

```
1: Let $x_k^{LP}$ represent the LP-relaxed solution of item $k$
2: Set $S_j \leftarrow 0, \forall j \in 1, ..., n$
3: for $j = 1$ to $n$ do
4:     if $x_j^{LP} \geq R_j$ then
5:         $S_j \leftarrow 1$
6:     end if
7: end for
8: return $S$
```

5.3 Experimental setup

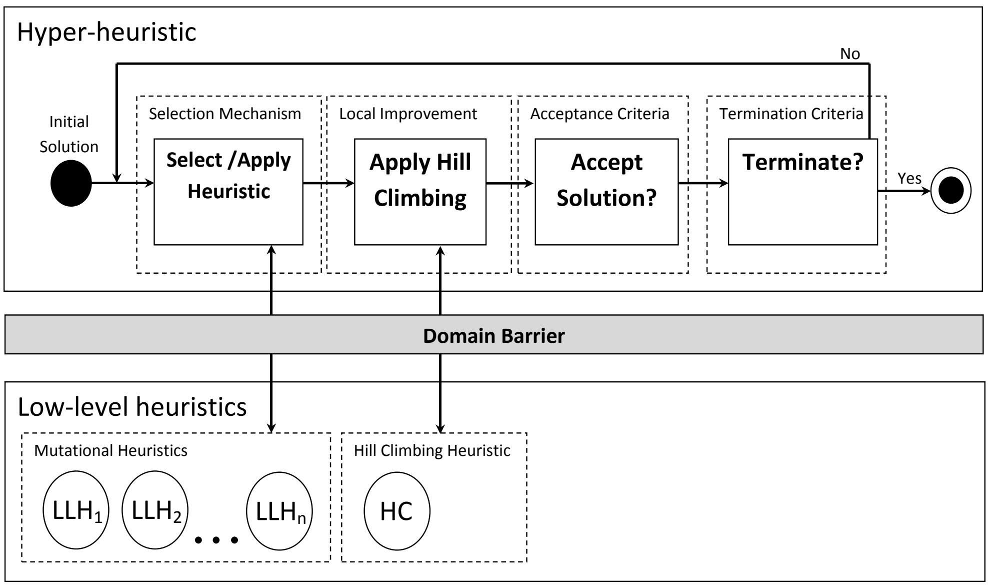
This section introduces the hyper-heuristic components tested and the test framework used. As with Chapter 4, the multidimensional knapsack problem will be used as a test problem domain for the methods defined in this chapter.

5.3.1 Hyper-heuristic framework

As discussed in Section 2.3.3, Özcan et al. [174] described four different hyper-heuristic frameworks. One of these frameworks F_C, selects and applies a mutational heuristic $LLH_i \in LLH_1, ..., LLH_n$, where n is the number of mutational heuristics available, followed by a pre-defined hill climber HC before deciding whether to accept the new solution. Such a framework is illustrated in Figure 5.2 and is the framework used in this chapter. Özcan et al. [174] observed that using F_C could yield better results than the traditional F_A framework on a number of benchmark functions.

Hyper-heuristics using an F_C framework have similar characteristics to Memetic Algorithms [159, 160]. Memetic Algorithms combine evolutionary algorithms and local search techniques. A simple Memetic Algorithm will attempt to improve each candidate solution in a population through some hill climbing mechanism. Memetic Algorithms have previously been shown to be successful on a number of different problem domains including the MKP [40, 170]. In this chapter the effect of introducing crossover into an F_C hyper-heuristic framework is analysed.

Figure 5.2: Single-point search hyper-heuristic framework with local improvement (F_C)



5.3.2 Hyper-heuristic components

Three heuristic selection methods and three acceptance criteria previously defined in Section 4.2 are used in this chapter, resulting in a total of nine hyper-heuristics to be tested. *Simple Random*, *Choice Function* and *Reinforcement Learning* heuristic selection are paired with each of *Only Improving*, *Simulated Annealing* and *Late Acceptance Strategy* move acceptance. These hyper-heuristics will operate over a set of seven low-level heuristics, previously introduced in Chapter 4: one-point crossover, two-point crossover, uniform crossover, Swap Dimension mutation and three variants of Parameterised Mutation. In the case of a crossover operator, two solutions are generated each time a low-level heuristic of this type is selected, with the best of the two solutions kept for consideration by the move acceptance criterion. The crossover and mutational low-level heuristics used are identical to those defined in Section 4.2.2. In the case of Parameterised Mutation, this operator is used with three different values to provide light, medium and heavy mutation. The mutation rates used are 10% (PARA10), 25% (PARA25) and 50% (PARA50) of the bits in a given solution respectively. In addition, a hill climbing heuristic is applied following each low-level heuristic invocation as defined by the F_C hyper-heuristic framework. The hill climber is a new low-level heuristic introduced in this chapter and is described below.

Hill climbing heuristic: In the existing literature [40, 180, 142] *add* and *drop* phases are often used in heuristics to improve or repair MKP solutions. A *utility-weight* is used to sort objects in order of their relative perceived value. Chu and Beasley [40] adopted the *surrogate duality* suggested by Pirkul [180] multiplying each weight by a relevance value r:

$$util_j = \frac{p_j}{\sum_{i=1}^{m} r_i w_{ij}} \tag{5.3.1}$$

Relevance values r_i are taken as the *dual variables* of each dimension i in the solution to the LP-relaxation of the MKP. As with the MKPHC low-level heuristic from Chapter 4, a local search operator for the MKP is implemented using relevance values. If a solution is infeasible, *drop* items from the knapsack in order of increasing *utility-weight* until it is feasible solution. When a solution is feasible, attempt to *add* items in order of decreasing *utility-weight* until a feasible solution cannot be found by adding another of the unselected items. Puchinger et al. [181] tested a number of efficiency methods, with the relevance weights of Chu and Beasley [40] based on dual variable values observed to be the best efficiency measure for the MKP. This heuristic is used as a local search mechanism after each crossover or mutational operator is applied, to repair and locally improve solutions as required by the F_C framework.

5.3.3 Experimental data and test framework definitions

As with the previous chapter, a run terminates after 10^6 fitness evaluations for each problem instance in order to directly compare results with the techniques in the literature. All hyper-heuristic experiments were carried out on an Intel Core 2 Duo 3 GHz CPU with 2 GB memory.

The three benchmark datasets introduced in Section 3.4 are used to test the proposed hyper-heuristics. The instances used for each experiment are identified in the relevant sections. Initial solutions are set as a single random binary string of length n, where n is the total number of objects associated with each instance. For tests using the SAC-94 benchmark set, a single run of each hyper-heuristic is sufficient as these instances are extremely small. In the case of the OR-Lib benchmark each set of 10 instances is taken from same distribution. As a result, taking the average %-gap over these 10 instances for each of the 27 sets effectively shows the performance of 10 runs of each hyper-heuristic. For the larger GK instances, each of the experiments are repeated 10 times to account for the stochastic nature of the hyper-heuristics with average performance over 10 runs reported.

For hyper-heuristics using *Simulated Annealing* move acceptance criterion the same parameters are used as in Chapter 4. The starting temperature is set as the distance between the fitness value of the initial solution and the value of the LP-relaxed solution of the current instance. As the run progresses the temperature is decreased linearly, until it reaches 0 at the end of the run. In *Late Acceptance Strategy* hyper-heuristics, a list length of 500 is used as suggested by Burke and Bykov [20] and Özcan et al. [175].

In all experiments using Memory Crossover, a memory size of $0.1 * n$ is used where n is the number of variables in the instance currently being solved. This will ensure that poor quality solutions which are found early in the search are removed from the list quickly, in favour of better quality solutions found later on.

5.4 Finding a suitable initialisation method for the list of solutions

Some preliminary experiments are required in order to validate the new initialisation method proposed in Section 5.2.2. The three initialisation techniques described in Section 5.2.2 (C*, R* and $jqdInit$) are tested on a subset of 90 instances of ORLib where $m \in \{5\}$ and $n \in \{100, 250, 500\}$ using the best performing hyper-heuristic from the previous experiments, *Choice Function - Only Improving*. Again the hyper-heuristic is allowed to run for 10^6 fitness evaluations on each instance. Table 5.1 details the average of the best solutions in the list for each initialisation method and Table 5.2 shows the average solution quality of all solutions in the list over each set of 10 instances. Standard deviations are given as subscripts. This gives some insight into the differing nature of the populations produced when using each initialisation method. The average best solution and average list quality when using R* is far superior to C*. This is unsurprising as R* was designed to generate solutions closer to the optimal solution than those generated with C*. The best solutions produced by *jqdInit* are also superior to C* on average. For some instances in this dataset *jqdInit* does not produce any feasible solutions. The best solutions produced by *jqdInit* are only slightly poorer in quality on average than those produced by R*. As *jqdInit* allows infeasible solutions, the average list quality is very poor in terms of fitness score and is significantly worse than both C* and R* for all datasets.

Table 5.1: Average best solutions for C*, R* and *jqdInit* initialisation methods over each set of 10 instances in the 90 ORLib instances with $m = 5$

Instance set	C*	R*	*jqdInit*
OR5x100-0.25	$19105_{2.31}$	$23948_{0.37}$	$16325_{0.57}$
OR5x100-0.50	$37136_{1.91}$	$43015_{0.26}$	$42742_{0.69}$
OR5x100-0.75	$55909_{0.86}$	$60158_{0.23}$	$60082_{0.33}$
OR5x250-0.25	$47840_{1.19}$	$60137_{0.16}$	$59902_{0.32}$
OR5x250-0.50	$94016_{0.51}$	$109080_{0.10}$	$108653_{0.24}$
OR5x250-0.75	$140632_{0.44}$	$151344_{0.06}$	$151255_{0.10}$
OR5x500-0.25	$94431_{0.86}$	$120392_{0.05}$	$119937_{0.27}$
OR5x500-0.50	$188748_{0.65}$	$219323_{0.03}$	$218962_{0.11}$
OR5x500-0.75	$280437_{0.35}$	$302185_{0.02}$	$301870_{0.05}$

Table 5.3 shows the results in terms of %-gap as the average over 10 instances for each instance type and standard deviations included as subscripts. Interestingly on these larger instances C* is the poorest performing initialisation method with an average %-gap of 0.67. Using *jqdInit* yields the best results over these instances achieving an average %-gap of 0.39, slightly outperforming R* which has an average %-gap of 0.43. Despite both the average and best solutions produced by the R* initialisation method being better than the *jqdInit* in all of the datasets tested, the new initialisation method leads to better results overall after a full hyper-heuristic run.

The key difference between the previous methods and the proposed initialisation method is the tolerance of infeasible solutions. These solutions may still contain the 'building blocks' of good quality solutions. The final solution quality does not seem to

Table 5.2: Average list quality for C*, R* and *jqdInit* initialisation methods over each set of 10 instances in the 90 ORLib instances with m = 5

Instance set	C*	R*	*jqdInit*
OR5x100-0.25	17781 $_{1.88}$	23545 $_{0.29}$	-64285 $_{63.77}$
OR5x100-0.50	35553 $_{1.31}$	42575 $_{0.38}$	-97229 $_{65.86}$
OR5x100-0.75	54633 $_{0.75}$	59777 $_{0.22}$	-184816 $_{73.04}$
OR5x250-0.25	45045 $_{1.07}$	59739 $_{0.15}$	-181532 $_{53.70}$
OR5x250-0.50	90814 $_{0.39}$	108657 $_{0.11}$	-284952 $_{73.26}$
OR5x250-0.75	137421 $_{0.32}$	150905 $_{0.08}$	-436705 $_{81.19}$
OR5x500-0.25	90016 $_{0.75}$	119951 $_{0.06}$	-348724 $_{60.51}$
OR5x500-0.50	183289 $_{0.26}$	218853 $_{0.03}$	-620003 $_{52.80}$
OR5x500-0.75	275380 $_{0.19}$	301674 $_{0.01}$	-918566 $_{62.05}$

Table 5.3: Performance of initialisation methods over the 90 ORLib instances with m = 5

Instance set	C*	R*	*jqdInit*
OR5x100-0.25	1.31 $_{0.17}$	1.48 $_{0.26}$	1.25 $_{0.23}$
OR5x100-0.50	0.63 $_{0.10}$	0.63 $_{0.16}$	0.62 $_{0.12}$
OR5x100-0.75	0.39 $_{0.07}$	0.38 $_{0.11}$	0.42 $_{0.08}$
OR5x250-0.25	0.70 $_{0.15}$	0.51 $_{0.11}$	0.45 $_{0.10}$
OR5x250-0.50	0.37 $_{0.09}$	0.26 $_{0.07}$	0.22 $_{0.04}$
OR5x250-0.75	0.25 $_{0.06}$	0.15 $_{0.04}$	0.15 $_{0.04}$
OR5x500-0.25	0.70 $_{0.12}$	0.25 $_{0.05}$	0.24 $_{0.04}$
OR5x500-0.50	1.19 $_{0.32}$	0.12 $_{0.03}$	0.13 $_{0.03}$
OR5x500-0.75	0.50 $_{0.18}$	0.08 $_{0.02}$	0.07 $_{0.01}$
Average	0.67 $_{0.14}$	0.43 $_{0.09}$	0.39 $_{0.08}$

be adversely affected as a result of this as seen in Table 5.3. This suggests that infeasible solutions can help the search process when solving the MKP, particularly as optimal solutions are known to be close to the boundary of feasibility. As *jqdInit* is competitive with the two existing methods from the literature it is used during all further experimentation.

5.5 Experiments

Experiments are performed controlling crossover at both the hyper-heuristic level and the domain level. In each case, the hyper-heuristics are initially tuned and tested over the ORLib benchmark set before their general applicability is assessed on two further datasets.

5.5.1 Controlling crossover at the hyper-heuristic level for the MKP

As described in Section 5.2 crossover can be controlled at the hyper-heuristic level with no domain-specific knowledge. When a second individual is required for crossover it is selected from a list of potential solutions maintained by the hyper-heuristic. To assess the impact of controlling crossover at the hyper-heuristic level in this framework, the experiments are performed for three separate test cases: with Random Crossover, with Memory Crossover and without crossover. Table 5.4 shows the performance of each hyper-heuristic over all ORLib instances using each of the crossover management strategies, with standard deviations included as subscripts. Due to space limitations acronyms are used for each *heuristic selection method - move acceptance criterion* combination.

Table 5.4: Average %-gap over all ORLib instances for each hyper-heuristic with Random Crossover, Memory Crossover and without crossover

Hyper-heuristic	Random Crossover	Memory Crossover	No Crossover
SR-OI	$1.16_{0.84}$	$1.12_{0.81}$	$1.11_{0.82}$
CF-OI	$1.18_{0.83}$	$1.19_{0.86}$	$\mathbf{1.07}_{0.80}$
RL-OI	$1.16_{0.81}$	$1.14_{0.84}$	$1.10_{0.84}$
SR-LAS	$2.79_{2.12}$	$1.20_{0.93}$	$2.54_{1.84}$
CF-LAS	$2.86_{2.19}$	$1.23_{0.97}$	$2.72_{2.01}$
RL-LAS	$2.67_{1.97}$	$1.20_{0.92}$	$2.48_{1.77}$
SR-SA	$2.35_{1.33}$	$1.21_{0.85}$	$2.10_{1.18}$
CF-SA	$2.30_{1.29}$	$1.19_{0.82}$	$2.10_{1.19}$
RL-SA	$2.21_{1.22}$	$1.21_{0.86}$	$2.04_{1.10}$

The best performing hyper-heuristic is *Choice Function - Only Improving* without crossover with the lowest average %-gap of 1.07 over all ORLib instances. Using *Only Improving* acceptance criterion is clearly superior on average to both *Late Acceptance Strategy* and *Simulated Annealing* in this framework, when no crossover or Random Crossover is used. In the case of *Only Improving* acceptance, all three crossover types perform similarly however when using *Late Acceptance Strategy* and *Simulated Annealing* as move acceptance criteria, the performance is much better if Memory Crossover is used. The results obtained using these hyper-heuristics (*Late Acceptance Strategy* and

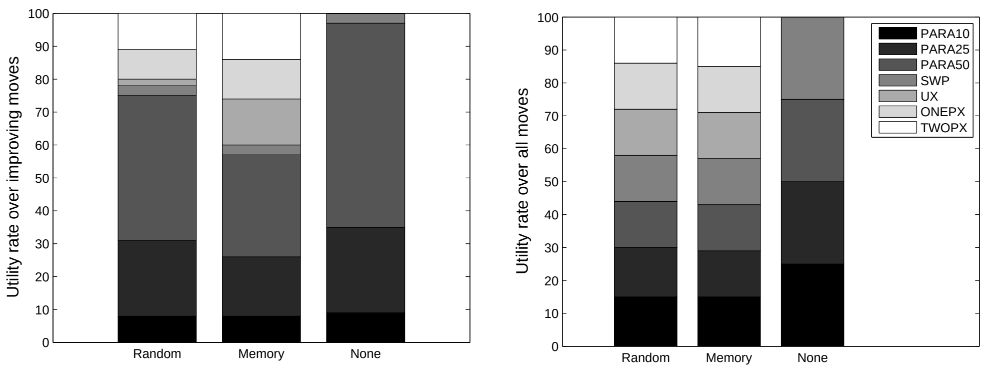
Simulated Annealing with Memory Crossover) do not vary significantly from the hyper-heuristics using *Only Improving* acceptance criterion.

Overall the %-gaps of the hyper-heuristics without crossover are lower than those that use Random Crossover suggesting that using crossover as a mutation operator in this way does not benefit the search. This supports previous assertions that the search space of heuristics can be reduced in an attempt to improve performance. Özcan and Basaran [170] noted that reducing the number of hill climbers (memes) can improve the performance of a Memetic Algorithm solving the MKP. Chakhlevitch and Cowling [37] also showed similar improvement when reducing the number of low-level heuristics in a hyper-heuristic framework operating on a scheduling problem. For each acceptance criterion there is little difference in the results obtained by using a different heuristic selection mechanism. However, there is significant difference between the results obtained using different acceptance criteria. This suggests that the move acceptance criterion used has a more significant impact on the performance of a hyper-heuristic than heuristic selection mechanism using this heuristic set. This behaviour was also observed by Özcan et al. [174] where a number of hyper-heuristics were tested over a set of benchmark functions.

Figure 5.3 shows the utilisation rates of each low-level heuristic for each of the *Choice Function - Only Improving* hyper-heuristics with Random Crossover, Memory Crossover and No Crossover (the best performing hyper-heuristic on average). *Utility rate* indicates the percentage usage of a low-level heuristic during a run. Figure 5.3(a) shows the utility rate of each heuristic considering only moves which improve on the current best-of-run solution. Figure 5.3(b) shows the average utility rate of each heuristic considering all moves (i.e. how many times each heuristic was chosen during the search process). These utility rates are average values over a single run of each instance over all 270 instances in ORLib. In all cases there are clearly stronger low-level heuristics on average however this is not reflected in the amount of times each heuristic is selected overall. Due to the nature of the *Choice Function*, some low-level heuristics will be selected at a higher rate than others at certain points of the search, usually through repeated invocation. Although in percentage terms this is roughly uniform over the full benchmark dataset, it is not the case that low-level heuristic selection is uniform for a particular instance. Moreover these figures show that all of the low-level heuristics available are capable of contributing to the improvement of a solution at a given stage for at least some of the instances. This provides a justification for their continued presence in the low-level heuristic set. Similar behaviour was observed for all hyper-heuristics tested.

Figure 5.3: Average low-level heuristic utilisation for *Choice Function - Only Improving* hyper-heuristics with Random, Memory and No Crossover over all instances in ORLib

(a) Utility rate over improving moves **(b)** Utility rate over all moves



5.5.2 Controlling crossover at the domain level for the MKP

Although benefit has been observed in some hyper-heuristics when including crossover as a mutation operator in this framework, the quality of solutions used to perform crossover could not be guaranteed. When the solutions used for crossover are taken to be best found solutions, crossover can benefit the hyper-heuristic framework. As discussed in Section 3.3, the constraints of the 0-1 multidimensional knapsack problem can be relaxed to take fractional values, yielding the related LP-relaxed version of the problem. It is known that the solutions to the LP-relaxed version of the MKP can provide good approximations for the 0-1 version of the problem [40]. Moreover, the framework developed for this work already performs the computational effort required to obtain the solutions to the LP-relaxed problem. Using the initialisation method for the list of solutions obtained in Section 5.4, the same nine hyper-heuristics are again applied to ORLib using the same parameters as before. Table 5.5 shows their performance in terms of %-gap over a single run of each instance of ORLib.

Table 5.5: Average %-gap over all ORLib instances for each hyper-heuristic using a list of solutions to provide the second input for crossover managed at the domain level

Heuristic Selection	Acceptance Criteria		
Mechanism	Only Improving	Late Acceptance	Simulated Annealing
Simple Random	0.74 $_{0.76}$	0.71 $_{0.74}$	0.71 $_{0.76}$
Choice Function	0.75 $_{0.78}$	**0.70** $_{0.74}$	0.71 $_{0.76}$
Reinforcement Learning	0.73 $_{0.74}$	0.71 $_{0.75}$	**0.70** $_{0.76}$

The best average %-gap over all ORLib instances is 0.70, obtained by *Choice Function - Late Acceptance Strategy* and *Reinforcement Learning - Simulated Annealing* hyper-heuristics. Following confirmation that the data is not normally distributed using a Shapiro-Wilk test, a Mann-Whitney U test within a 95% confidence interval shows no statistically significant difference between these two hyper-heuristics. Interestingly, those hyper-heuristics with *Late Acceptance* and *Simulated Annealing* acceptance outperform those with *Only Improving* acceptance. This is in contrast to the previous hyper-heuristics, which controlled crossover at the hyper-heuristic level, where *Only Improving* acceptance performed best. This is closer to what would be expected, as *Simulated Annealing* and *Late Acceptance Strategy* are designed to overcome the problem of becoming trapped in local optima. Despite this, no clear conclusions can be drawn as to why this reversal of performance is observed in the case that crossover is controlled at the domain-specific level. As with the previous experiments, the acceptance criterion used has a greater effect on the quality of solutions obtained than heuristic selection method. Although there are two 'best' performing hyper-heuristics within this framework, only one hyper-heuristic from each framework will be compared in the following section. The *Choice Function - Late Acceptance* is taken to compare to the best performing hyper-heuristic from Section 5.5.1 and CPLEX 12.5.

5.5.3 Comparison of hyper-heuristics managing crossover at the hyper-heuristic level, hyper-heuristics managing crossover at the domain level and CPLEX 12.5

Table 5.6 shows detailed results for each instance type for *Choice Function - Late Acceptance Strategy* with crossover controlled at the domain-specific level, the best performing hyper-heuristic from Section 5.5.1 (*Choice Function - Only Improving* with No Crossover) and results obtained using CPLEX 12.5. CPLEX 12.5 is allowed to run for a maximum of 1800 CPU seconds per instance, with a maximum working memory of 8GB. For reference, the hyper-heuristics in the previous sections perform 10^6 fitness evaluations in 25-30 seconds on average per instance.

Table 5.6: Detailed performance of CPLEX 12.5, *Choice Function - Late Acceptance Strategy* with crossover managed at domain level and *Choice Function - Only Improving* with No Crossover on ORLib instances based on average %-gap)

Problem Set	CPLEX 12.5	CF-LAS	CF-OI$_{noXO}$
OR5x100-0.25	0.99 $_{0.19}$	1.16 $_{0.20}$	1.22 $_{0.25}$
OR5x100-0.50	0.45 $_{0.09}$	0.53 $_{0.08}$	0.59 $_{0.16}$
OR5x100-0.75	0.32 $_{0.07}$	0.40 $_{0.07}$	0.39 $_{0.08}$
OR5x250-0.25	0.22 $_{0.04}$	0.42 $_{0.04}$	0.51 $_{0.10}$
OR5x250-0.50	0.11 $_{0.02}$	0.20 $_{0.03}$	0.42 $_{0.19}$
OR5x250-0.75	0.08 $_{0.01}$	0.13 $_{0.01}$	0.21 $_{0.04}$
OR5x500-0.25	0.07 $_{0.01}$	0.19 $_{0.03}$	0.60 $_{0.13}$
OR5x500-0.50	0.04 $_{0.01}$	0.10 $_{0.03}$	0.85 $_{0.13}$
OR5x500-0.75	0.02 $_{0.00}$	0.06 $_{0.01}$	0.32 $_{0.09}$
OR10x100-0.25	1.56 $_{0.20}$	2.00 $_{0.22}$	2.08 $_{0.37}$
OR10x100-0.50	0.79 $_{0.10}$	1.02 $_{0.19}$	1.16 $_{0.15}$
OR10x100-0.75	0.48 $_{0.07}$	0.58 $_{0.08}$	0.66 $_{0.06}$
OR10x250-0.25	0.45 $_{0.07}$	0.83 $_{0.09}$	1.02 $_{0.18}$
OR10x250-0.50	0.23 $_{0.02}$	0.39 $_{0.06}$	0.58 $_{0.11}$
OR10x250-0.75	0.14 $_{0.02}$	0.23 $_{0.03}$	0.41 $_{0.06}$
OR10x500-0.25	0.18 $_{0.02}$	0.40 $_{0.06}$	1.10 $_{0.35}$
OR10x500-0.50	0.08 $_{0.01}$	0.18 $_{0.02}$	1.20 $_{0.31}$
OR10x500-0.75	0.06 $_{0.01}$	0.12 $_{0.01}$	0.61 $_{0.16}$
OR30x100-0.25	2.89 $_{0.28}$	3.45 $_{0.46}$	3.91 $_{0.57}$
OR30x100-0.50	1.32 $_{0.14}$	1.56 $_{0.26}$	1.85 $_{0.27}$
OR30x100-0.75	0.82 $_{0.06}$	0.92 $_{0.08}$	1.04 $_{0.20}$
OR30x250-0.25	1.05 $_{0.07}$	1.55 $_{0.17}$	2.12 $_{0.25}$
OR30x250-0.50	0.48 $_{0.03}$	0.71 $_{0.08}$	1.08 $_{0.14}$
OR30x250-0.75	0.28 $_{0.02}$	0.39 $_{0.04}$	0.52 $_{0.08}$
OR30x500-0.25	0.50 $_{0.05}$	0.92 $_{0.10}$	1.99 $_{0.27}$
OR30x500-0.50	0.22 $_{0.02}$	0.39 $_{0.05}$	1.66 $_{0.10}$
OR30x500-0.75	0.14 $_{0.01}$	0.23 $_{0.02}$	0.82 $_{0.15}$
Average$_{StdDev}$	0.52 $_{0.06}$	0.70 $_{0.09}$	1.07 $_{0.18}$

Table 5.7 compares each technique in terms of %-gap against one another for statistical significance (t-test, 95% confidence), indicating the number of instance sets in which there is a particular variation in performance. For two techniques A_1 and A_2, $\geq$ indicates that A_1 outperforms A_2 on average whilst $\gg$ indicates that this difference is

statistically significant (conversely $\leq$ and $\ll$). When comparing the performance of the best performing hyper-heuristic from Section 5.5.1 (*Choice Function - Only Improving* with No Crossover) and the best performing hyper-heuristic with crossover controlled at the domain-specific level (*Choice Function - Late Acceptance Strategy*), controlling crossover at the domain-specific level results in better performance on average for 26 of the 27 sets of instances. This difference is statistically significant in 22 of these cases. CPLEX 12.5 statistically significantly outperforms both hyper-heuristics in all 27 sets of instances.

Table 5.7: Number of instance sets particular statistical differences occur

A_1 *vs* A_2	$\ll$	$\leq$	$\geq$	$\gg$
CF-LAS *vs* CF-OI$_{noXO}$	0	1	4	22
CF-LAS *vs* CPLEX 12.5	27	0	0	0
CF-OI$_{noXO}$ *vs* CPLEX 12.5	27	0	0	0

The general applicability of the best performing hyper-heuristic with crossover controlled at the domain-specific level is tested by applying it to two further benchmark sets, each with differing properties. SAC-94 is a set of benchmark instances from classic papers in the literature using mostly real-world data (as described in Section 5.3.3), with optimal solutions known for all problems. It is difficult to perform a direct comparison with techniques over these instances due to the difference in termination criterion and running times. For example, some methods in the literature provide the best results over 30 runs or more. If an algorithm finds the optimal solution in at least 5% of trial runs for a given instance it is deemed a successful run. The success rate over each dataset is therefore the number of successful runs divided by the number of problems in the set. A *Choice Function - Late Acceptance Strategy* hyper-heuristic performs a single run on each instance as before. Table 5.8(a) shows the performance of the hyper-heuristic in terms of success rate over each set of instances in SAC-94. The *Choice Function - Late Acceptance Strategy* hyper-heuristic with crossover controlled at the domain-specific level performs at least as well as *Choice Function - Only Improving* with No Crossover in every group of instances in this set.

The final benchmark set on which to test the hyper-heuristics is the set of 11 large instances provided by Glover and Kochenberger [88]. The results for both hyper-heuristics are given as the average of 10 runs on each of these instances. Table 5.8(b) shows the performance of these hyper-heuristic over these large instances and also presents results for CPLEX 12.5. The LP-relaxed optimal solutions are again used as a basis to derive %-gap. Standard deviations of the average %-gaps over all 11 instances are included as subscripts. The *Choice Function - Only Improving* hyper-heuristic with

No Crossover performs relatively badly on this larger set of instances, obtaining an average %-gap of 0.92 compared to 0.45 obtained by the *Choice Function - Late Acceptance Strategy* hyper-heuristic with crossover controlled at the domain-specific level. CPLEX 12.5 obtains an average %-gap of 0.21.

Table 5.8: (a) Success rate of CPLEX 12.5, *Choice Function - Late Acceptance Strategy* hyper-heuristic and *Choice Function - Only Improving* with No Crossover over all SAC-94 instances and (b) Performance of CPLEX 12.5, *Choice Function - Late Acceptance Strategy* hyper-heuristic and *Choice Function - Only Improving* with No Crossover on Glover and Kochenberger instances

(a)

Dataset	Num of Instances	CPLEX 12.5	CF-LAS	CF-OI$_{noXO}$
hp	2	1.00	0.00	0.00
pb	6	1.00	0.67	0.50
pet	6	1.00	0.50	0.34
sento	2	1.00	1.00	1.00
weing	8	1.00	0.63	0.63
weish	30	1.00	1.00	0.64

(b)

Instance	LPopt Score	CPLEX 12.5 %-gap	CF-LAS %-gap	CF-OI$_{noXO}$%-gap
GK01	3776	0.26	0.57 $_{1.49}$	1.33 $_{6.82}$
GK02	3976	0.46	0.81 $_{3.86}$	1.60 $_{9.66}$
GK03	5671	0.26	0.63 $_{3.10}$	1.64 $_{18.25}$
GK04	5794	0.46	0.91 $_{3.77}$	1.84 $_{18.18}$
GK05	7577	0.23	0.45 $_{3.00}$	0.83 $_{13.61}$
GK06	7703	0.34	0.76 $_{5.02}$	1.54 $_{23.00}$
GK07	19232	0.08	0.19 $_{6.48}$	0.33 $_{18.81}$
GK08	18831	0.15	0.33 $_{5.68}$	0.55 $_{9.57}$
GK09	58101	0.02	0.07 $_{7.47}$	0.10 $_{12.95}$
GK10	57318	0.05	0.14 $_{8.68}$	0.16 $_{14.07}$
GK11	95278	0.05	0.13 $_{12.34}$	0.15 $_{15.10}$
Average	-	0.21	0.45$_{0.30}$	0.92$_{0.69}$

Comparison to previous approaches

Table 5.9 shows the results of the best hyper-heuristic presented in this chapter, *Choice Function - Late Acceptance Strategy* with crossover controlled at the domain-specific level and CPLEX 12.5, compared to a number of heuristic and metaheuristic techniques from the literature. From this table it can be seen that the hyper-heuristics presented here perform well in comparison to many previous approaches. The use of 10^6 fitness evaluations as a termination criterion allows direct comparison to previous meta-heuristic approaches [184, 40, 170, 101]. The %-gap of 0.70 obtained by the hyper-heuristic is better than the previous metaheuristic methods of Özcan and Basaran [170] and Hinterding [101] and a number of existing heuristic methods. The best %-gap is still obtained by the Memetic Algorithm of Chu and Beasley [40] and the variant of their work provided by Raidl [184]. The %-gap over all instances of ORLib of 0.52 obtained by CPLEX 12.5 is unsurprisingly far better than that of 3.14 achieved by Chu and Beasley [40] using CPLEX 4.0, offering similar performance to the best metaheuristic techniques from the literature. Although a direct comparison can not be performed due to differing termination criteria, there has clearly been a marked improvement in the performance of exact solvers such as CPLEX in recent years.

Table 5.9: Average %-gap of other (meta-)heuristics and CPLEX over all instances in ORLib

Type	Reference	%-gap
MIP	**CPLEX 12.5**	**0.52**
MA	Raidl [184]	0.53
MA	Chu and Beasley [40]	0.54
Hyper-heuristic	**CF-LAS**	**0.70**
MA	Özcan and Basaran [170]	0.92
Permutation GA	Hinterding [101] (Raidl [184])	1.30
Heuristic	Pirkul [180]	1.37
Heuristic	Freville and Plateau [73]	1.91
Heuristic	Qian and Ding [183]	2.28
MIP	Chu and Beasley [40] (CPLEX 4.0)	3.14
Heuristic	Akçay et al. [2]	3.46
Heuristic	Volgenant and Zoon [208]	6.98
Heuristic	Magazine and Oguz [142]	7.69

The currently best known results in literature for the ORLib instances were obtained by Vasquez and Vimont [206]. Results from this study are only available for the largest instances of ORLib where $n = 500$. Results for these instances obtained with a *Choice Function - Late Acceptance Strategy* hyper-heuristic and CPLEX 12.5 are provided along with the results of Vasquez and Vimont [206] for comparison in Table 5.10, again with standard deviation as subscripts. *Choice Function - Late Acceptance Strategy* is chosen arbitrarily for comparison from the best performing hyper-heuristics for the previous experiments. The standard deviations for the hyper-heuristic and CPLEX 12.5 are available in Table 5.6.

Table 5.10: Performance comparison with best metaheuristic technique in the literature over ORLib instances with n = 500 objects.

	Vasquez and Vimont [206]		**CPLEX 12.5**		**CF-LAS**	
Instance	%-gap	t[s]*	%-gap	t[s]	%-gap	t[s]
OR5x500-0.25	0.07 $_{0.01}$	14651*	0.07 $_{0.01}$	315	0.19 $_{0.03}$	11
OR5x500-0.50	0.04 $_{0.05}$	6133*	0.04 $_{0.01}$	141	0.10 $_{0.03}$	16
OR5x500-0.75	0.02 $_{0.00}$	7680*	0.02 $_{0.00}$	46	0.06 $_{0.01}$	22
OR10x500-0.25	0.17 $_{0.02}$	10791*	0.19 $_{0.02}$	534	0.40 $_{0.06}$	14
OR10x500-0.50	0.08 $_{0.00}$	8128*	0.09 $_{0.01}$	675	0.18 $_{0.02}$	21
OR10x500-0.75	0.06 $_{0.01}$	6530*	0.06 $_{0.01}$	567	0.12 $_{0.01}$	29
OR30x500-0.25	0.48 $_{0.05}$	30010*	0.53 $_{0.05}$	1800	0.92 $_{0.10}$	23
OR30x500-0.50	0.21 $_{0.02}$	35006*	0.23 $_{0.02}$	1800	0.39 $_{0.05}$	39
OR30x500-0.75	0.14 $_{0.01}$	45240*	0.14 $_{0.01}$	1800	0.23 $_{0.02}$	55
Average$_{StdDev}$	0.14 $_{0.02}$	18241*	**0.15** $_{0.02}$	**853**	**0.29** $_{0.03}$	**26**

The results of *Choice Function - Late Acceptance Strategy* are obtained in a fraction of the time taken by Vasquez and Vimont and are less than 0.15% closer to the LP-relaxed optimum in absolute terms. Again a Mann-Whitney U test within a 95% confidence interval is performed in order to assess statistical significance. For each set of 10 instances in Table 5.10, there is no statistically significant difference in performance between *Choice Function - Late Acceptance Strategy* and the method of Vasquez and Vimont. CPLEX 12.5 offers performance comparable on average to Vasquez and Vimont [206] with no statistically significant difference between these results.

A fundamental goal of hyper-heuristic research is to provide solutions which are 'good enough, soon enough, cheap enough' [21]. Although the work of Vasquez and Vimont [206] was performed using inferior hardware there is a stark contrast in execution times of each technique[1]. The results of CPLEX 12.5 are obtained in significantly less time

[1]Note on CPU times based on Dongarra [60]:

- Intel P4 1700 MHz = 796 MFLOP/s
- Intel P4 2 GHz (estimated) 796 * 2 / 1.7 = 936.47 (scaled from 1.7 GHz to 2GHz)
- Intel Core 2 Q6600 Kensfield (1 core) 2.4 GHz = 2426 MFLOP/s
- Intel Core 2 Duo 3 GHz (estimated) 2426 * 3 / 2.4 = 3032.5 MFLOP/s (scaled from 2.4 to 3 GHz)

Based on the above Intel Core 2 Duo 3 GHz is estimated 3032.5 / 936.47 = 3.24 times faster. t[s]* for Vasquez and Vimont [206] in Table 5.10 are scaled using these CPU times.

than Vasquez and Vimont [206] but the average time is far greater than the average time of 25.6 seconds per instance taken by *Choice Function - Late Acceptance Strategy*. CPLEX 12.5 is run for a maximum of 1800 seconds on each instance however only takes an average of 853 seconds an instance. This average is lower than the maximum time allowed as a run will finish early if an optimal solution is found or constraints on memory are exceeded. Puchinger et al. [182] provided results for CPLEX 9.0 over these 90 instances attaining an average %-gap of 0.16. Overall CPLEX 12.5 achieves an average %-gap of 0.15 and equals or outperforms the previous results in every set except one (OR10x500-0.25). This discrepancy may be attributed to differing parameter settings within CPLEX or human error when collating results.

An indirect comparison between techniques can be made on a subset of the instances in SAC-94 in terms of success rate as shown in Table 5.11. Three common problem instance sets from SAC-94 are used for comparison, the pet problem set (with pet2 omitted), the sento problem set and the last two instances of the weing problem set. The Memetic Algorithm of Chu and Beasley [40] performs particularly well with Particle Swarm Optimisation and Grammatical Evolution performing particularly badly. The *Choice Function - Late Acceptance Strategy* hyper-heuristic performs well in comparison to the results in the literature. For completeness, CPLEX 12.5 is also tested and finds optimal solutions for entire SAC-94 dataset using the hardware and settings outlined previously taking a maximum of 0.3 seconds per instance.

Table 5.11: Success rate of techniques from the literature over a subset SAC-94 instances

Technique	Reference	sento	pet	weing
MIP	**CPLEX 12.5**	**1.00**	**1.00**	**1.00**
Memetic Algorithm	Chu and Beasley [40]	1.00	1.00	1.00
Memetic Algorithm	Cotta and Troya [46]	1.00	1.00	1.00
Multimeme Memetic Algorithm	Özcan and Basaran [170]	1.00	0.80	0.50
Hyper-heuristic	**CF-LAS**	**1.00**	**0.60**	**0.50**
Attribute Grammar	Cleary and O'Neill [42]	0.50	0.80	0.50
Genetic Algorithm	Khuri et al. [123]	0.50	0.60	0.50
Particle Swarm Optimisation	Hembecker et al. [100]	0.00	-	0.50
Grammatical Evolution	Cleary and O'Neill [42]	0.00	0.20	0.00

Table 5.12 compares the performance of *Choice Function - Late Acceptance Strategy* and CPLEX 12.5 with the methods of Raidl and Gottlieb [185], using the benchmarks provided by Glover and Kochenberger [88]. Raidl and Gottlieb [185] experimented with a number of different representations in evolutionary algorithms for the MKP. The three

best results were obtained from direct representation (DI), weight-biased representation (WB) and permutation representation (PE). The results of their study are taken as averages over 30 runs and were allowed 10^6 non-duplicate individuals. Standard deviations for the 30 runs of each instance by Raidl and Gottlieb [185] are provided as subscripts. Each hyper-heuristic tested is also allowed 10^6 evaluations however duplicate individuals are counted. The direct encoding of Raidl and Gottlieb [185] outperforms *Choice Function - Late Acceptance Strategy* however the hyper-heuristic compares favourably to the other two encoding methods shown. Although only an indirect comparison can be made due to the differing nature of each techniques termination criterion and subsequently running times, CPLEX performs particularly well on these instances with an average %-gap of 0.21 compared to the 0.45 %-gap of the *Choice Function - Late Acceptance Strategy* hyper-heuristic.

Table 5.12: Performance comparison of *Choice Function - Late Acceptance Strategy* hyper-heuristic, CPLEX 12.5 and permutation, weight-biased and direct representation evolutionary algorithms on Glover and Kochenberger instances

	CPLEX	DI	**CF-LAS**	WB	PE
Instance	%-gap	%-gap	%-gap	%-gap	%-gap
GK01	0.26	$0.27_{0.03}$	$0.57_{0.04}$	$0.31_{0.08}$	$0.38_{0.07}$
GK02	0.45	$0.46_{0.01}$	$0.81_{0.10}$	$0.48_{0.05}$	$0.50_{0.06}$
GK03	0.26	$0.37_{0.01}$	$0.63_{0.05}$	$0.45_{0.04}$	$0.52_{0.06}$
GK04	0.47	$0.53_{0.02}$	$0.91_{0.07}$	$0.67_{0.08}$	$0.71_{0.09}$
GK05	0.21	$0.29_{0.00}$	$0.45_{0.04}$	$0.40_{0.05}$	$0.46_{0.07}$
GK06	0.32	$0.43_{0.02}$	$0.76_{0.07}$	$0.61_{0.06}$	$0.70_{0.09}$
GK07	0.06	$0.09_{0.00}$	$0.19_{0.03}$	$0.38_{0.08}$	$0.52_{0.09}$
GK08	0.14	$0.17_{0.01}$	$0.33_{0.03}$	$0.53_{0.07}$	$0.75_{0.09}$
GK09	0.02	$0.03_{0.00}$	$0.07_{0.01}$	$0.56_{0.04}$	$0.89_{0.08}$
GK10	0.04	$0.05_{0.00}$	$0.14_{0.02}$	$0.73_{0.07}$	$1.10_{0.07}$
GK11	0.05	$0.05_{0.00}$	$0.13_{0.01}$	$0.87_{0.06}$	$1.24_{0.06}$
Average	$\mathbf{0.21}_{0.16}$	$0.25_{0.18}$	$\mathbf{0.45}_{0.30}$	$0.54_{0.17}$	$0.71_{0.27}$

5.6 Framework generality tested on the Boolean satisfiability problem

As the hyper-heuristics developed in this chapter have been implemented in a generic way, they can be applied to any problem domain which can be represented as a binary string. One such domain is the well-studied Boolean satisfiability problem (SAT). Given a number of decision variables and associated clauses, the objective of SAT is to assign the variables in such a way that each clause is satisfied and subsequently the entire formula evaluates to true. The performance of the best hyper-heuristic from the previous section will be compared to two existing hyper-heuristic approaches from the literature which have been applied to this problem. Although not strictly as relevant to a second problem domain, the 'domain-specific' aspects of the hyper-heuristic framework can still be used. SAT is chosen as a second test problem as the LP-relaxed version of the problem can also be used in the initialisation process in the same way as the MKP.

The low-level heuristics and initialisation methods used are identical to those used in the MKP. The only two components changed are the fitness function and the hill climbing heuristic used after each application of a low-level heuristic. The hill climber flips a random variable in each unsatisfied clause and keeps the change if an improvement is made. All instances tested are from the 'Uniform Random-3-SAT' category and are taken from SATLIB [104]. All of these instances are known to be satisfiable. A single run of the best performing hyper-heuristic from Section 5.5, *Choice Function - Late Acceptance Strategy* is performed with 5 seconds allowed for each instance with the results presented in Table 5.13. The %-gap measure is again used to measure the performance of a given algorithm, with a lower value indicating that a solution is closer to the optimal. As all instances are satisfiable an optimal solution exists in this case the %-gap value is calculated as:

$$100 * \frac{OptimalSolution - SolutionFound}{OptimalSolution} \tag{5.6.1}$$

Where $OptimalSolution$ is the fitness value of the optimal solution to a given problem and $SolutionFound$ is the fitness value of a solution obtained by a particular method.

Table 5.13: Performance of a *Choice Function - Late Acceptance Strategy* hyper-heuristic over SATLIB benchmarks

Test-Set	Instances	Variables	Clauses	Average Clauses Satisfied	%-gap
uf20-91	1000	20	91	91	0
uf150-645	100	150	645	637.34	1.19
uf200-860	100	200	860	848.09	1.38
uf250-1065	100	250	1065	1049.75	1.43

The hyper-heuristic implemented finds a satisfied solution for all 100 instances in the uf20-91 set in a single run. Swan et al. [200] used a subset of these instances to test a number of different hyper-heuristics. In their study the best hyper-heuristic tested found the optimal solution 10.35% of the time on 100 runs on the first 20 instances of this set. In terms of average number of clauses left unsatisfied, the best hyper-heuristic of Swan et al. [200] leaves on average 3.06 clauses unsatisfied on these instances, in contrast *Choice Function - Late Acceptance Strategy* leaves no clauses unsatisfied for all instances. As no termination criterion or running time is stated by Swan et al. [200], it is difficult to provide a direct comparison and comment on the general performance of this method.

Hyde et al. [109] used the uf250-1065 instances to test a number of hyper-heuristics with *Simple Random* heuristic selection and move acceptance criteria generated using Genetic Programming. In this work, the evolved acceptance criteria were able to outperform both *All Moves* and *Great Deluge* acceptance criteria on these instances. The best results obtained over 95 of the instances in this set (five were used for training purposes) obtained an average %-gap obtained of 5.595% over 30 runs of the Genetic Programming hyper-heuristic. *Choice Function - Late Acceptance Strategy* obtains an average %-gap of 1.43 for these instances.

5.7 Concluding remarks

Two frameworks for controlling crossover in single-point selection hyper-heuristics are proposed, using a common NP-hard combinatorial optimisation problem as a test bed. Crossover is included at two levels, firstly it is controlled at the hyper-heuristic level where no domain-specific information is used, secondly it is controlled below the domain barrier and given domain-specific information. In each case a list of potential second solutions to be used in crossover is maintained. Perhaps unsurprisingly, crossover performs better when it is controlled below the domain barrier and problem-specific information is available. Although the management of crossover is desirable at the domain-level in this case, unfortunately it is not always possible to access domain-level information in other hyper-heuristic frameworks. This raises questions regarding the definition of hyper-heuristics and exactly where the responsibility of managing the inputs for low-level heuristics should lie. Additional questions also emerge regarding the notion of state within low-level heuristics. In general selection hyper-heuristics typically operate over a set of stateless low-level heuristics. In the case of hyper-heuristic level crossover control, it could be argued that managing the input for crossover low-level heuristics in such a way adds stateful behaviour. Whether this fits in to the original definition of hyper-heuristics is open to interpretation and a case could be made both for and against this argument. Exact solvers such as CPLEX also make use of heuristics and have improved significantly since Chu and Beasley [40] presented their results using CPLEX 4.0. Improved results using CPLEX 12.5 are included over the benchmark libraries for the use of future researchers in this area.

The hyper-heuristics in this chapter have operated over low-level heuristics using a simple binary representation. Therefore they can be applied to any problem domain which uses this representation. It has been shown that by simply changing the fitness evaluation function and the hill climbing low-level heuristic required by an F_C hyper-heuristic framework, it is possible to outperform existing hyper-heuristic techniques on instances of the Boolean satisfiability problem (SAT).

The next chapter shifts the focus of the thesis to cross-domain optimisation. The core goal of cross-domain optimisation is to develop methods which are general enough to perform well over a range of benchmark problem domains. Increasingly, such frameworks include crossover operators in the set of low-level heuristics. There are some issues when using some traditional hyper-heuristic components over cross-domain optimisation benchmarks, particularly with *Choice Function* heuristic selection. A modified variant of the traditional *Choice Function* heuristic selection method is proposed, with the intention of dealing with some of the limitations of the *Choice Function* when operating in a cross-domain environment.

CHAPTER 6

An Improved *Choice Function* Heuristic Selection for Cross-Domain Heuristic Search

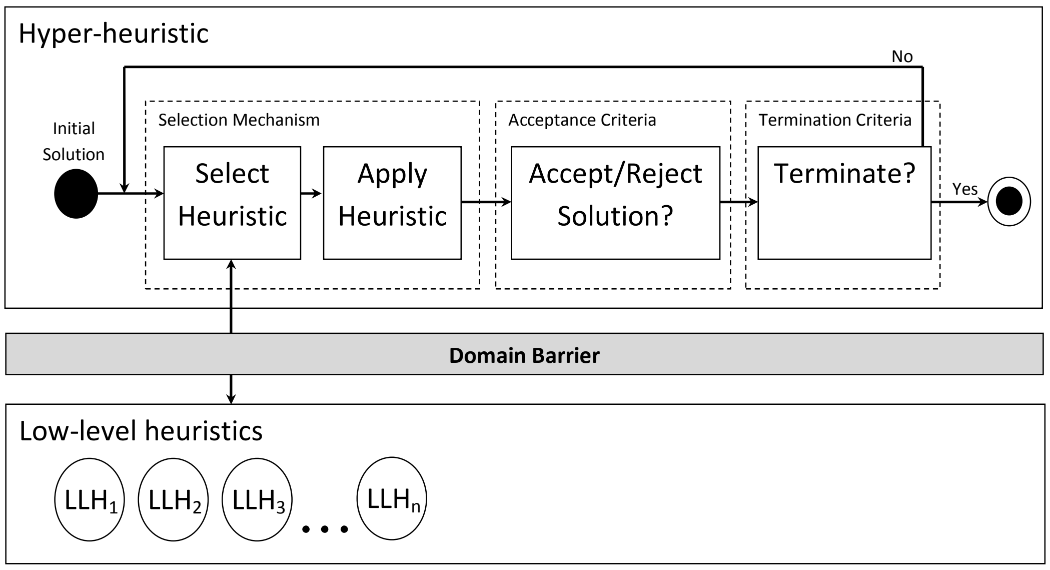
Chapter 5 investigated the use of crossover at two conceptual levels in selection hyper-heuristic frameworks. Crossover low-level heuristics are increasingly included in modern selection hyper-heuristics. One such framework is HyFlex [27, 166], developed to support the first Cross-domain Heuristic Search Challenge, CHeSC2011 [165]. This framework provides a common interface enabling easy comparison of cross-domain heuristic search methods.

Hyper-heuristics have been introduced in previous chapters as a class of high-level search technologies to solve computationally difficult problems, operating on a search space of low-level heuristics rather than solutions directly. A large number of heuristic selection methods exist for search hyper-heuristics. The *Choice Function* is an elegant heuristic selection method introduced in the very first hyper-heuristic paper for combinatorial optimisation presented by Cowling et al. [47]. Although the *Choice Function* has been shown to work well in a variety of different problem domains, the use of *Choice Function*-based hyper-heuristics is limited in cross-domain heuristic search using the CHeSC2011 benchmarks. This chapter presents the application of *Choice Function*-based hyper-heuristics to these benchmarks, highlighting the weakness of this heuristic selection method in the context of cross-domain heuristic search. A *Modified Choice Function* heuristic selection method is proposed, with the intensification and diversification parameters of the *Choice Function* controlled automatically using a method inspired by Reinforcement Learning. The proposed method is tested and compared to previous approaches over the six benchmark problem domains provided by HyFlex for CHeSC2011. Following this, the *Modified Choice Function* is tested on the MKP and compared to the best performing hyper-heuristic of Chapter 5 using *Late Acceptance Strategy* move acceptance criterion.

6.1 Selection hyper-heuristics and the *Choice Function*

Traditional single-point based search hyper-heuristics rely on two key components, a heuristic selection method and a move acceptance criteria as depicted in Figure 6.1. Using the ontology of Özcan et al. [174], previously discussed in Section 2.3.3, such a framework is classified as an F_A hyper-heuristic. Hyper-heuristics working within this framework operate on a single solution and repeatedly select and apply low-level heuristics to this solution. At each stage a decision is made as to whether to accept the move until some termination criterion is met.

Figure 6.1: Classic F_A single-point search hyper-heuristic framework



6.1.1 *Choice Function* heuristic selection

The *Choice Function* was introduced as a heuristic selection method by Cowling et al. [47] in the first hyper-heuristic paper in the field of combinatorial optimisation. The *Choice Function* scores heuristics based on a combination of three different measures and applies the heuristic with the highest rank at each given step. Each measure is weighted appropriately to provide balance between intensification and diversification during the heuristic search process. Choosing the right parameter values to weight these measures is not a trivial process and a small number of methods have been proposed in the literature. The mechanics of the *Choice Function* are introduced in detail in Section 2.3.3. Further to this work, Cowling et al. [48] described a method to adaptively manage these parameters. A mechanism is proposed which increases the weights of α or β following the application of a heuristic selected by the *Choice Function* which results in an improvement in the objective value. Although no specific implementation details are provided, this reward is said to be proportional to the size of improvement over the previous solution. Conversely, if a decrease in solution quality is obtained these weights are penalised proportionally to the change in objective value. Using this mechanism lead to an improved performance compared to the original results. The implementation of the *Choice Function* described and used by Bilgin et al. [14] for benchmark function optimisation and Özcan et al. [175] and Burke et al. [35] for examination timetabling is used in this chapter. This implementation increases α and β and reduces δ by the same value if an improvement is made and reduces α and β and increases δ if no improvement is made.

6.1.2 *Modified Choice Function* heuristic selection

Using the classic version of the *Choice Function* has some limitations when applied to cross-domain optimisation using the CHeSC2011 benchmarks. Firstly, the proportional improvement gained by a given heuristic is often not of interest but rather whether there has been any improvement at all. In the early stages of a search, a relatively poor heuristic can gain a large reward if it obtains a large improvement in objective value from a poor starting position. Later on in a search, a heuristic may yield a small improvement which is much more significant in the context of the optimisation process but will not receive such a large reward for this improvement. Secondly, if no improving solutions are found for a period of time, the *Choice Function* can very quickly descend into random search if the weighting is dominated by the diversification component. This can be a useful trait, however the rate at which the diversification increases in significance must be controlled. Özcan et al. [175] observed that *Simple Random* heuristic selection with Late Acceptance Strategy move acceptance performed very well on a set of examination timetabling instances. In this particular case very few perturbative low-level heuristics were implemented.

Reinforcement Learning [115] is a general term for a well-studied family of machine learning techniques, often used in its own right as a heuristic selection mechanism (introduced in Section 2.3.3). In selection hyper-heuristics, Reinforcement Learning techniques are often used in an online context, deciding which action to take from a set of possible actions based on previous experience. Nareyek [162] investigated using *Rein-*

forcement Learning as a heuristic selection method when solving constraint satisfaction problems (CSP). In such a heuristic selection mechanism, each low-level heuristic is assigned a utility value. At each step of the search process a heuristic is selected and applied based on these utility values. If the application of the selected low-level heuristic is considered successful, the utility value for that particular heuristic is increased, improving its chances of selection at the next iteration. Conversely if the heuristic application is considered unsuccessful it is decreased at the next iteration.

Here a *Modified Choice Function* (MCF) heuristic selection method is proposed, which aims to address the issues of using the *Choice Function* for cross-domain optimisation. In the *Modified Choice Function*, the parameters weighting each of f_1, f_2 and f_3 in the *Choice Function* are managed through a method inspired by Reinforcement Learning. This mechanism relies on a system of reward and punishment in order to tune these parameters. As with the original *Choice Function* of Cowling et al. [47], the *Modified Choice Function* does not make a distinction between the values of α or β which weight f_1 and f_2 respectively. For clarity these values are considered as a single intensification parameter which is referred to as ϕ. Also in line with the original *Choice Function*, this value is also used to weight each previous invocation of a given heuristic, giving greater importance to recent performance. The parameter to weight f_3 is used to control the level of diversification of heuristic search as before and will still be referred to as δ. In the *Modified Choice Function* the score F_t for each heuristic h_j is defined as:

$$F_t(h_j) = \phi_t f_1(h_j) + \phi_t f_2(h_k, h_j) + \delta_t f_3(h_j) \qquad (6.1.1)$$

where t is the current step in the search process. At each stage if an improvement in objective value is made, ϕ is rewarded and set to a static maximum value close to the upper limit of the interval (0,1) whilst δ is concurrently reduced to a static minimum value close to the bottom end of this interval. This leads to a greater emphasis on intensification and greatly reduces the level of diversity in heuristic selection choice each time an improvement is obtained. If no improvement in objective value is made, the level of intensification is decreased by linearly, reducing ϕ and the weighting of diversification is increased at the same rate through the δ parameter. This gives the intensification component of the *Choice Function* more time as the dominating factor in the calculation of F. For the experiments in this chapter the parameters ϕ_t and δ_t are defined as:

$$\phi_t(h_j) = \begin{cases} 0.99, & \text{if an improving move is made} \\ \max\{\phi_{t-1} - 0.01, 0.01\}, & \text{if a non-improving move is made} \end{cases} \qquad (6.1.2)$$

$$\delta_t(h_j) \ = 1 - \phi_t(h_j) \tag{6.1.3}$$

The use of 0.99 as the maximum value and 0.01 as the minimum value, ensure that the diversification component of the *Modified Choice Function* always has some non-negative influence on the F value for each heuristic.

6.2 Experimentation

A number of experiments are performed, using the CHeSC2011 benchmarks, to compare the performance of the original *Choice Function* with the proposed *Modified Choice Function* heuristic selection methods. Section 6.2.1 describes the experimental settings used for the hyper-heuristics tested. Section 6.2.2 compares the *Choice Function* and *Modified Choice Function* combined with *All Moves* move acceptance to a set of 'default' hyper-heuristics provided by the organisers of CHeSC2011 prior to the competition. The two *Choice Function - All Moves* variants are compared to the 20 CHeSC2011 entrants in Section 6.2.3 before a more direct comparison is provided in Section 6.2.4.

6.2.1 Experimental setup

In order to isolate the behaviour of the heuristic selection method component of each hyper-heuristic, the two *Choice Function* variants are paired with simple *All Moves* move acceptance criterion. This is a reasonable choice as Cowling et al. [47] highlighted *Choice Function - All Moves* as a good *heuristic selection method - move acceptance criterion* combination. A 'run' is considered as a notional 10 computational minutes, calculated using a benchmark provided by the CHeSC2011 organisers. In order for a fair comparison to be made, crossover heuristics are ignored as the original *Choice Function* provides no details of how to manage operators which require more than one solution as input. Methods to manage the second inputs for crossover have been introduced in Chapter 5 and will be used in the context of cross-domain optimisation in Chapter 7.

6.2.2 Relative performance of *Choice Function - All Moves* and *Modified Choice Function - All Moves* hyper-heuristics compared to CHeSC2011 'default' hyper-heuristics

Prior to the CHeSC2011 competition, the results of eight default hyper-heuristics were provided by the organisers to assess the performance of a hyper-heuristic [165]. These hyper-heuristics were inspired by state-of-the-art techniques. Each hyper-heuristic performs a single run on 10 instances for each of four problem domains; Boolean satisfiability (MAX-SAT), one-dimensional bin packing, personnel scheduling and permutation flow shop. They are then ranked using a system based on the Formula One scoring system, the best performing hyper-heuristic for each instance is awarded 10 points, the second 8 points and then each further hyper-heuristic awarded 6, 5, 4, 3, 2, 1 and 0 points respectively. Where there are more than 8 competitors, all hyper-heuristics below 8th position are awarded 0 points. As there are 40 instances tested with a maximum score of 10 for each instance, the maximum possible score a hyper-heuristics can obtain is 400. This scoring system is discussed in detail in Section 2.5.3. As this ranking system is based on relative performance, the *Choice Function* and *Modified Choice Function* are compared to the competition entries independently.

Table 6.1(a) shows the results of the *Modified Choice Function* heuristic selection combined with *All Moves* move acceptance criterion (MCF-AM) compared with the eight default hyper-heuristics (HH1-HH8). Table 6.1(b) shows the results of the same experiments using the original *Choice Function* and *All Moves* acceptance (CF-AM) as implemented by Bilgin et al. [14] and Burke et al. [35].

These tables show that the *Modified Choice Function - All Moves* hyper-heuristic outperforms all of the CHeSC default hyper-heuristics in MAX-SAT and personnel scheduling. More importantly the *Modified Choice Function* outperforms the original *Choice Function* in all four problem domains, although both versions seem to struggle more on the bin packing and permutation flow shop instances. This could be due to the omission of crossover operators if such operators perform well in these problem domain. The best performing hyper-heuristic in this set (HH4) is based on Iterated Local Search.

Table 6.1: Formula One scores for (a) a single run of *Modified Choice Function - All Moves* hyper-heuristic and the CHeSC default hyper-heuristics and (b) a single run of classic *Choice Function - All Moves* hyper-heuristic and the CHeSC default hyper-heuristics

(a)

	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8	*MCF-AM*
MAX-SAT	55.25	73.25	36.5	24.5	1	46	51.75	14.5	**87.25**
Bin Packing	59	61	**76**	71	15	51	39	1	17
Personnel Scheduling	64	57.5	22	50.5	50	0	49.5	31	**65.5**
Permutation Flow Shop	30	21	26.5	86	19.5	**77.5**	21	69	39.5
Overall	208.25	212.75	161	**232**	85.5	174.5	161.25	115.5	209.25

(b)

	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8	*CF-AM*
MAX-SAT	58	**78**	39.5	25.5	1	49	54.5	15.5	69
Bin Packing	59	61	**78**	71	19	51	38	7	6
Personnel Scheduling	**65.5**	64.5	23	52.5	53	0	52	31	48.5
Permutation Flow Shop	38	27.5	32	**86**	25.5	78.5	26.5	70	6
Overall	220.5	231	172.5	**235**	98.5	178.5	171	123.5	129.5

6.2.3 Relative performance of *Choice Function - All Moves* and *Modified Choice Function - All Moves* hyper-heuristics compared to CHeSC2011 entrants

Following the CHeSC2011 competition, the results were provided for the competition entries over a subset of the problems of all six problem domains, as introduced in Section 2.5.3. In total 30 instances are tested, with 5 instances being used from each of the six problem domains. Using the Formula One scoring system the maximum score obtainable for a single hyper-heuristic is 300. The CHeSC2011 competition results were taken as the median of 31 runs of each hyper-heuristic for each instance. Here results are also taken as the median of 31 runs in order to maintain consistency and allow direct comparison to the competition entries. Table 6.2(a) shows the results of the classic *Choice Function* and *All Moves* acceptance criterion compared to the 20 competition entries using the Formula One scoring system. Table 6.2(b) shows the results of the same experiments using *Modified Choice Function* heuristic selection and *All Moves* move acceptance criterion.

Using the classic *Choice Function* performs particularly badly against the other competition entries, ranking 20th out of 21 overall only obtaining a single point in personnel scheduling. Although overall this can be considered poor performance, it must be noted that for a given instance only the top 8 competitors are awarded points. This means that *Choice Function - All Moves* is in the top 8 for one instance of personnel scheduling. The *Modified Choice Function* scores 38.85 points in total, ranking 12th out of 21 hyper-heuristics. Since the competition results were made available, Gaspero and Urli [81] described variations of their original method (*AVEG-Nep*), which are also based on Reinforcement Learning. The best of the variants included in this paper ranked 13th overall compared to the original competitors. Here it is observed that managing the parameter settings of a *Choice Function* using Reinforcement Learning inspired techniques can outperform such methods, ranking 12th overall. For this hyper-heuristic points are only scored in two problem domains, personnel scheduling and MAX-SAT, leaving room for improvement in the other four domains. The vast majority of these points were scored in MAX-SAT (32.85) where *Modified Choice Function - All Moves* excels. When compared to the competition entries, the *Modified Choice Function - All Moves* outperforms all other competitors. It is likely that a very small number of heuristics are providing improvement in this problem domain and the increased focus on intensification is providing the gain in performance. The remaining points are all scored in the personnel scheduling domain. Figure 6.2 shows a breakdown of the number of points awarded to each technique over the MAX-SAT competition instances.

Table 6.2: Results of the median of 31 runs of (a) the classic *Choice Function - All Moves* hyper-heuristic and (b) the *Modified Choice Function - All Moves* hyper-heuristic, compared to CHeSC2011 competitors using Formula One scores

(a)

Rank	Name	Score
1	AdapHH	181
2	VNS-TW	134
3	ML	131.5
4	PHunter	93.25
5	EPH	89.25
6	HAHA	75.75
7	NAHH	75
8	ISEA	71
9	KSATS-HH	66
10	HAEA	53.5
11	ACO-HH	39
12	GenHive	36.5
13	DynILS	27
14	SA-ILS	24.25
15	XCJ	22.5
16	AVEG-Nep	21
17	GISS	16.75
18	SelfSearch	7
19	MCHH-S	4.75
20	**CF - AM**	**1**
21	Ant-Q	0

(b)

Rank	Name	Score
1	AdapHH	177.1
2	VNS-TW	131.6
3	ML	127.5
4	PHunter	90.25
5	EPH	88.75
6	NAHH	72.5
7	HAHA	71.85
8	ISEA	68.5
9	KSATS-HH	61.35
10	HAEA	52
11	ACO-HH	39
12	**MCF - AM**	**38.85**
13	GenHive	36.5
14	DynILS	27
15	SA-ILS	22.75
16	XCJ	20.5
17	AVEG-Nep	19.5
18	GISS	16.25
19	SelfSearch	5
20	MCHH-S	3.25
21	Ant-Q	0

Figure 6.2: Number of points scored in the MAX-SAT domain using the Formula One system for each CHeSC2011 competitor and *Modified Choice Function - All Moves* hyper-heuristic

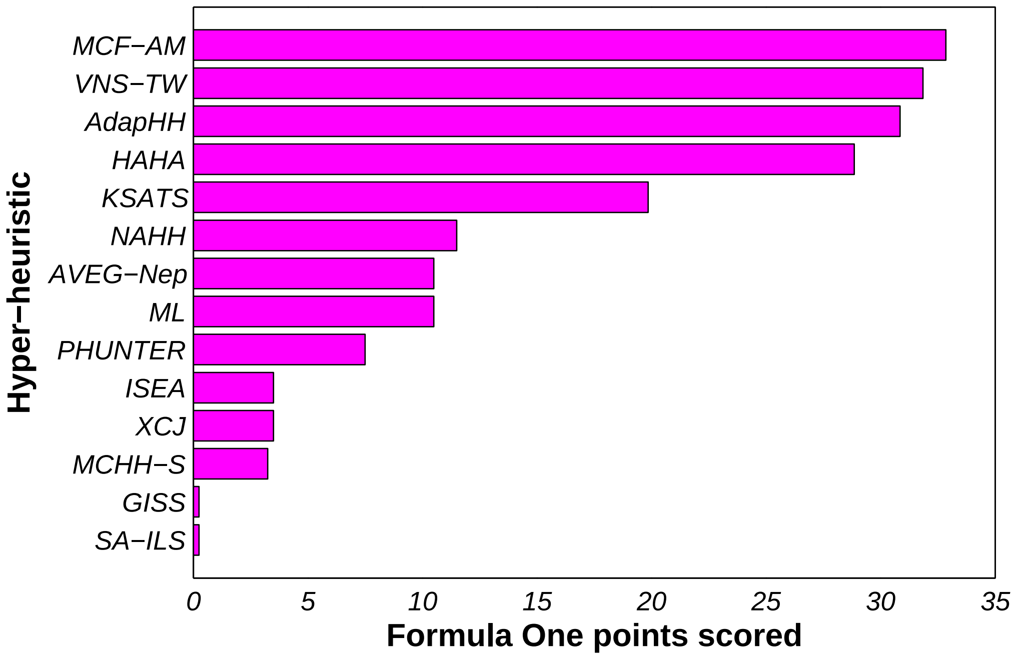


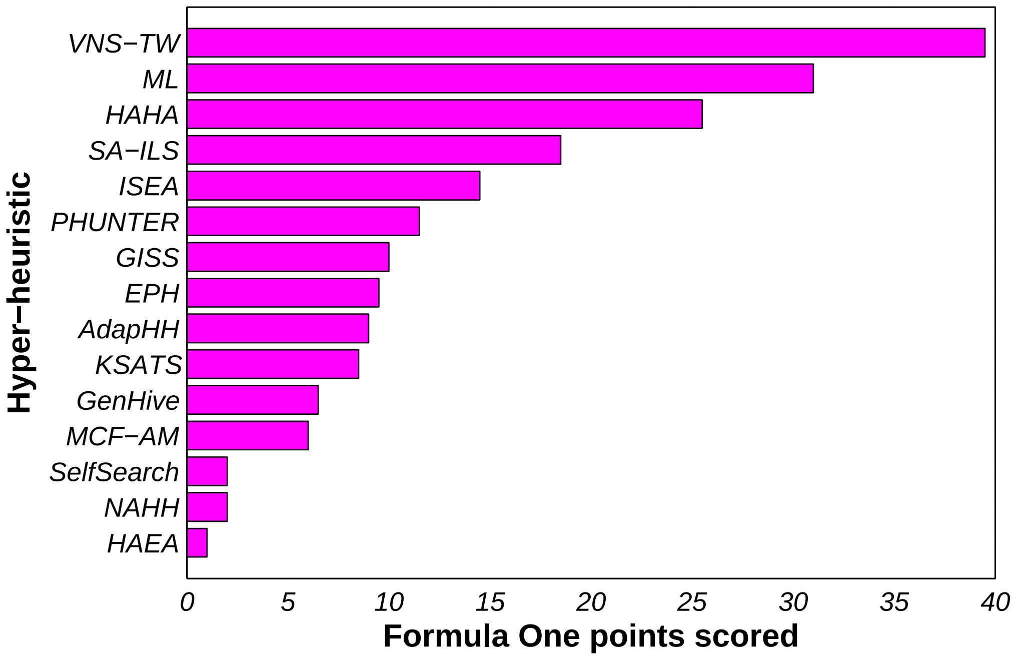
Figure 6.3 shows a breakdown of the number of points awarded to each technique over the personnel scheduling competition instances. Please note that hyper-heuristics which score zero points for these domains are omitted from these figures.

As with any ranking mechanism, there are issues with one method potentially gaining an advantage simply by the metrics of the comparison method used. Gaspero and Urli [81] used a normalised cost function value to compare the relative performance of hyper-heuristics. This can be generalised to compare hyper-heuristics over an arbitrary number of instances or domains. The median objective function value of the 31 runs for a given instance are normalised to a value $\in [0,1]$, using the maximum and minimum fitness value obtained for all hyper-heuristics. The normalised objective function value obj_{norm}, for a given problem instance $inst$ is calculated as:

$$obj_{norm}(inst) = \frac{obj_{actual}(inst) - obj_{best}(inst)}{obj_{worst}(inst) - obj_{best}(inst)} \tag{6.2.1}$$

where obj_{actual} represents the actual median objective achieved in this instance by a given algorithm and $obj_{best}(inst)$ and $obj_{worst}(inst)$ represent the best and worst median objective values obtained by any of the CHeSC2011 competitors. Figure 6.4 shows the normalised objective function values over all 30 instances for the 20 CHeSC2011 com-

Figure 6.3: Number of points scored in the personnel scheduling domain using the Formula One system for each CHeSC2011 competitor and *Modified Choice Function - All Moves* hyper-heuristic



petitors and *Choice Function - All Moves*. Figure 6.5 provides the same figure using the *Modified Choice Function - All Moves* and CHeSC2011 entrants. In these figures, the 21 hyper-heuristics being compared are sorted by median normalised objective function value with a lower value indicating better performance.

The box plots provided in Figure 6.4 and Figure 6.5 give an indication of relative variation in performance for each hyper-heuristic over all domains. It is clear from these figures that both *Choice Function - All Moves* and *Modified Choice Function - All Moves* do not provide consistent relative performance over the six benchmark problem domains used in CHeSC2011. Relative to the competition entrants, *Choice Function - All Moves* has the worst median normalised objective function value of all hyper-heuristics with *Modified Choice Function - All Moves* only faring slightly better, ranking 18th out of the 21 hyper-heuristics compared. In the case of the *Modified Choice Function - All Moves* hyper-heuristic, the lack of consistency in performance over all six domains is exaggerated by how well it performs in the MAX-SAT problem domain. This leads to a large distribution in normalised objective function value with only *DynILS* offering a greater variation in performance.

Figure 6.4: Box and whisker comparison of 20 CHeSC2011 entrants and *Choice Function - All Moves* using normalised objective function

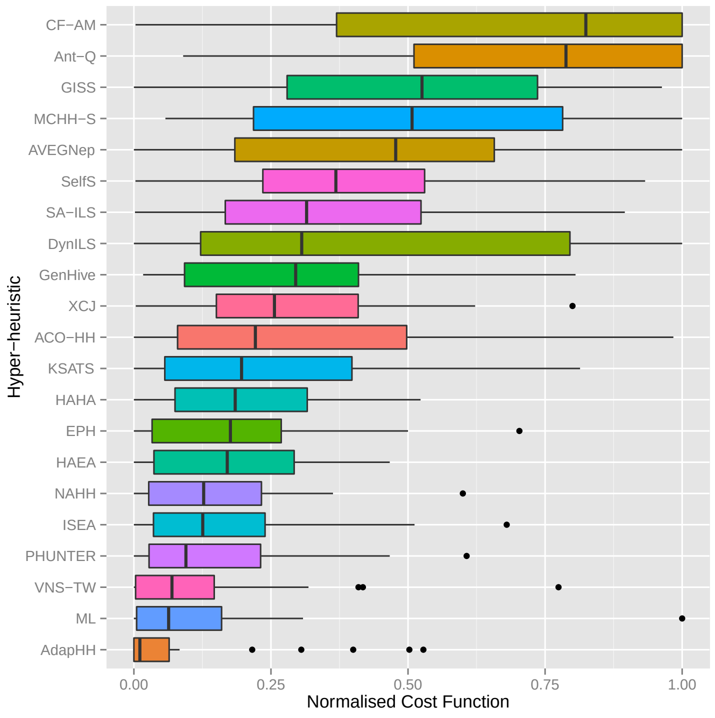
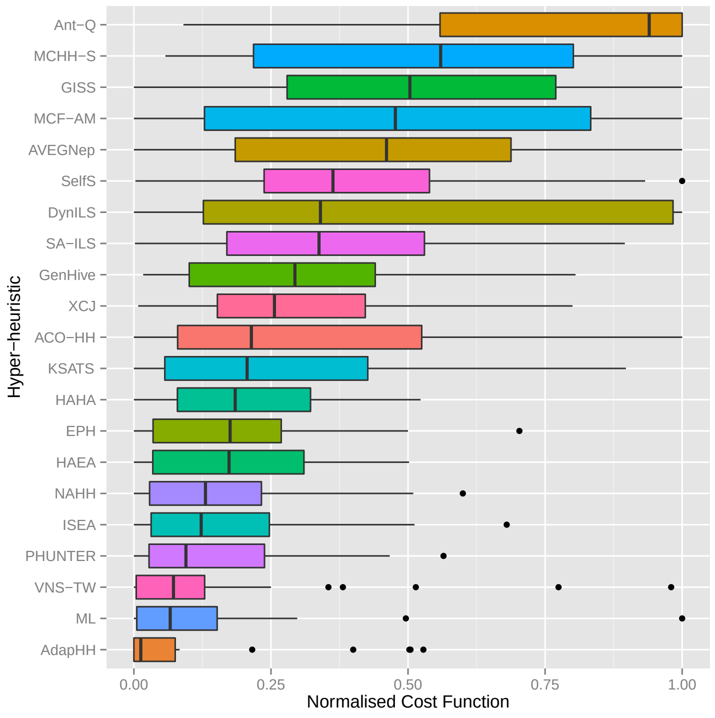


Figure 6.5: Box and whisker comparison of 20 CHeSC2011 entrants and *Modified Choice Function - All Moves* using normalised objective function



6.2.4 Direct comparison between *Choice Function - All Moves* and *Modified Choice Function - All Moves* hyper-heuristics using CHeSC2011 benchmark instances

The Formula One scoring system is limited in that it only measures relative performance against a set of previous approaches. A direct comparison between the two experiments can be performed on the raw objective values achieved by both hyper-heuristics. Table 6.3 shows the results of an independent Student's t-test within a 95% confidence interval on the values of each of the 31 runs for each instance in the competition. Where this table refers to a particular instance number, this represents the order in which this instance appears in the list of the instances tested for CHeSC2011 from Table 2.11, rather than the HyFlex index of that problem instance[1]. In this table, $\gg$ ($\ll$) denotes that using *Modified Choice Function - All Moves* (*Choice Function - All Moves*) is performing statistically significantly better than using *Choice Function - All Moves* (*Modified Choice Function - All Moves*). Additionally, > (<) denotes that there is no statistically significant performance variation between *Modified Choice Function - All Moves* and *Choice Function - All Moves* however *Modified Choice Function - All Moves* (*Choice Function - All Moves*) performs slightly better on average.

Table 6.3: Pairwise comparison between *Modified Choice Function - All Moves* and *Choice Function - All Moves* using an independent Student's t-test

Problem Domain	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5
MAX-SAT	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
Bin Packing	<	$\gg$	$\gg$	<	$\ll$
Personnel Scheduling	<	>	>	<	>
Permutation Flow Shop	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$
TSP	$\ll$	<	$\ll$	$\gg$	>
VRP	$\gg$	$\gg$	$\gg$	$\gg$	$\gg$

These results show the *Modified Choice Function* statistically significantly outperforming the classic *Choice Function* in 3 of the 6 problem domains; MAX-SAT, permutation flow shop and the vehicle routing problem. In many cases there is no statistically significant difference in performance. In only 3 of the 30 problem instances does the original *Choice Function - All Moves* performs statistically significantly better than *Modified Choice Function - All Moves*. These three cases are interesting as they are in problem domains where it is possible for either hyper-heuristic to statistically significantly outperform the other on an individual instance.

[1]e.g. instance 1 for MAX-SAT is the first instance tested in this domain with HyFlex index 3 in Table 2.11

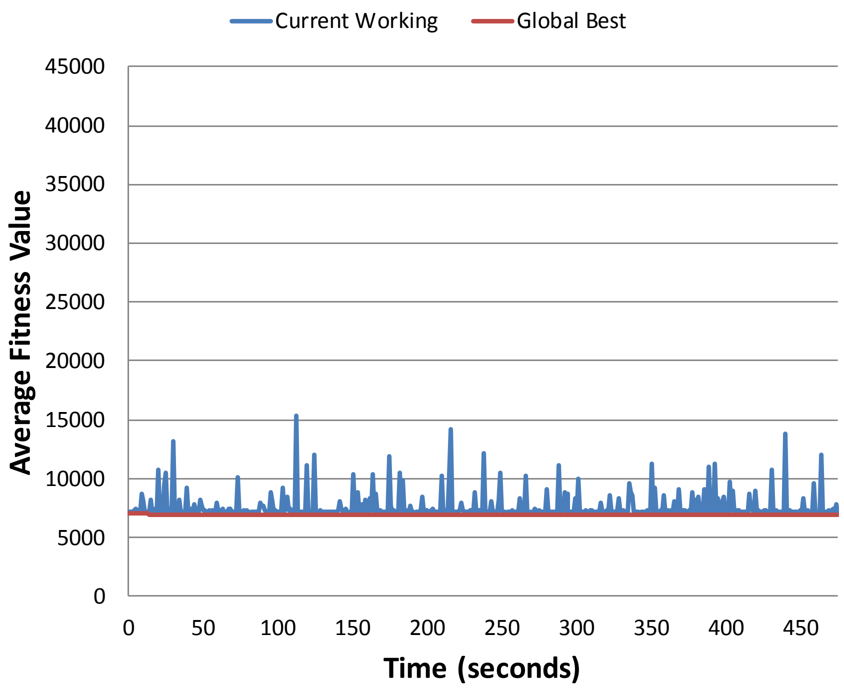
In order to analyse the performance of both hyper-heuristics to a greater level of depth, extra analysis can be performed on the performance during individual runs. Instance 3 and Instance 4 from the travelling salesman problem domain are chosen, as each hyper-heuristic statistically significantly outperforms the other in one of these instances. These instances correspond to 'rat575' and 'u2152' from Reinelt [187] with HyFlex indexes of 2 and 7, and consist of 575 and 2152 cities respectively. Figure 6.6 and Figure 6.7 plot the average fitness value of the current 'working' solution and best solution found so far over time, for 10 runs of a given hyper-heuristic. Each run lasts for 484 seconds, as allowed by the benchmarking tool provided by the CHeSC2011 organisers for the machine used. As the travelling salesman problem is modelled as a minimisation problem, a lower objective function value is better.

Figure 6.6(a) and Figure 6.6(b) show the average performance of 10 runs of *Choice Function - All Moves* and *Modified Choice Function - All Moves* on 'rat575'. In this instance, *Choice Function* was shown to statistically significantly outperform the *Modified Choice Function* in Table 6.3. The red line on each figure corresponds to the average fitness value of the best solution found so far, with the blue line denoting the average fitness value of the current working solution. Overall there is limited improvement in objective function value in both hyper-heuristics relative to the range of fitness values encountered. In the case of *Choice Function - All Moves*, the average working solution value is between around 7000 and 1500 during the search process. There is evidence of some diversification in solution quality, however in general the working value tends towards the best value encountered. With *Modified Choice Function - All Moves*, there is a high average current working solution value in the early stages of the search. As this method focusses on intensification, it is likely that in some runs this hyper-heuristic can get stuck in poor quality regions of the search space, particularly early in the search if given a poor starting point. After the initial stages where searching within poor quality regions is more prevalent, the average current working solution values for *Modified Choice Function - All Moves* are within a much smaller range than for *Choice Function - All Moves*. On this instance *Choice Function - All Moves* outperforms *Modified Choice Function - All Moves* in terms of best solution found at the end of a run, with each hyper-heuristic averaging scores of 6963.12 and 6979.59 respectively.

Figure 6.7(a) and Figure 6.7(b) show the average performance of 10 runs of *Choice Function - All Moves* and *Modified Choice Function - All Moves* on 'u2152'. In this instance, *Modified Choice Function* was seen to statistically significantly outperform the *Choice Function* in Table 6.3. Very similar behaviour is observed by both hyper-heuristics to that shown in Figure 6.6 for the previous instance. Again, it is possible for the *Mod-*

Figure 6.6: Average working and global best fitness values with respect to time for 10 runs on the 'rat575' instance of the travelling salesman problem

(a) *Choice Function - All Moves*



(b) *Modified Choice Function - All Moves*

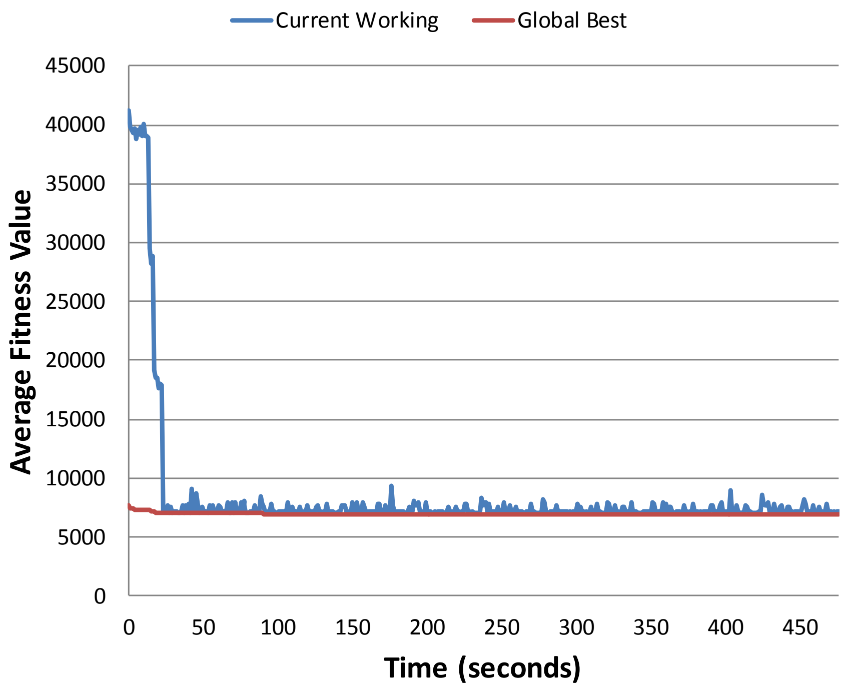
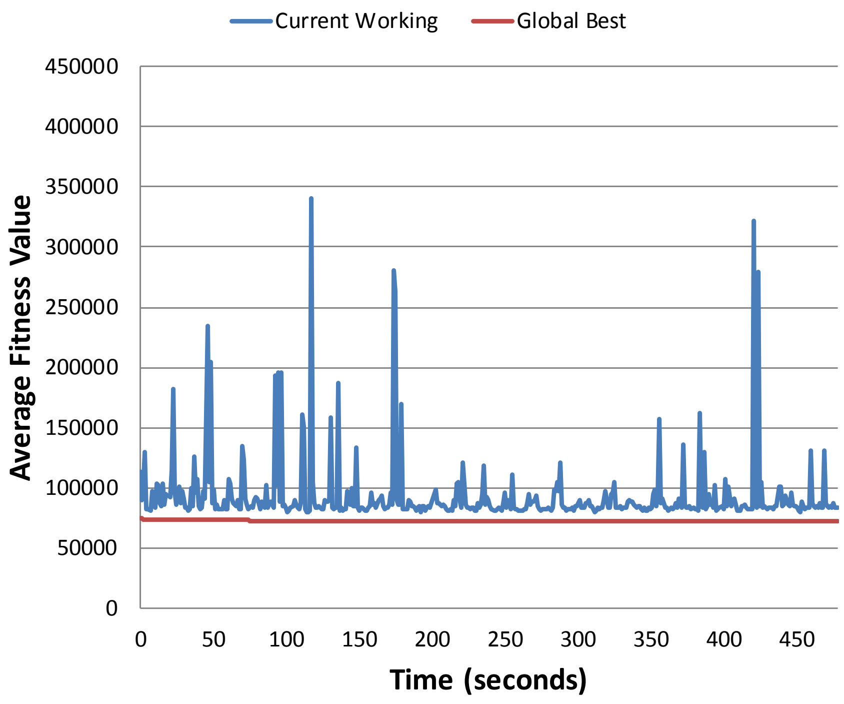
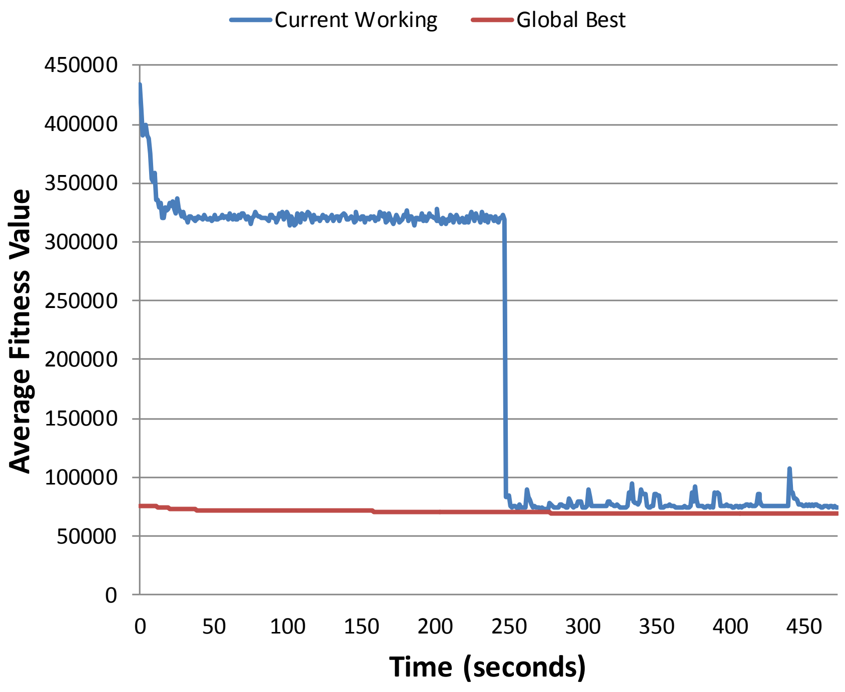


Figure 6.7: Average working and global best fitness values with respect to time for 10 runs on the 'u2152' instance of the travelling salesman problem

(a) *Choice Function - All Moves*



(b) *Modified Choice Function - All Moves*



ified Choice Function to be rooted in areas of the search space with poor quality solutions. Similar patterns also emerge in terms of the range of objective function values observed for the current working solution. *Choice Function - All Moves* again operates over a much larger range of objective function values once the effects of the early stages of the *Modified Choice Function* have worn off. In this case *Modified Choice Function - All Moves* outperforms *Choice Function - All Moves* in terms of best solution found with each hyper-heuristic averaging scores of 69731.92 and 72270.33 respectively.

Figure 6.6 and Figure 6.7 along with Table 6.3 highlight a key issue in hyper-heuristic design. Similar behaviour patterns are shown by both hyper-heuristics on two instances of the same type with varying results in terms of performance. In particular, the emphasis placed on intensification by the *Modified Choice Function* is beneficial in the case of 'u2152' but not 'rat575'. These results suggest that there are in fact properties on a per instance basis, as well as on a per domain basis, for which a particular hyper-heuristic will work well. This adds an extra layer of complexity when designing hyper-heuristics general enough to perform well on multiple domains, as good performance in a given instance of a problem does not necessarily suggest that a hyper-heuristic will work well for all instances of that domain.

6.3 Performance of the *Modified Choice Function* on the MKP

In Chapter 5 a number of hyper-heuristics utilising crossover were tested on the multidimensional knapsack problem (MKP). This section briefly compares the performance of the *Modified Choice Function* introduced in this chapter with the best performing hyper-heuristic from Chapter 5. The *Modified Choice Function* is paired with *Late Acceptance Strategy* acceptance criterion, in order to isolate the effect of varying the heuristic selection method. Exactly the same experimental setup is used from Chapter 5 using three well-known benchmark libraries from the literature. In each case a run for a single instance terminates once 10^6 fitness evaluations have been performed. A single run of each hyper-heuristic is performed on each of the instances in the ORLib and SAC-94 benchmark sets. In the case of the larger GK instances provided by Glover and Kochenberger [88], results are presented as the average of 10 runs for each instance. Crossover is controlled at the domain level, maintaining a list of elite solutions initialised using the *jqdInit* method (introduced in Section 5.2.2) to be used when a second solution is required for crossover. The only difference between the *Choice Function - Late Acceptance Strategy* hyper-heuristic of Chapter 5 and *Modified Choice Function - Late Acceptance Strategy* is the weighting of f_1, f_2 and f_3.

Table 6.4 shows the results of the *Modified Choice Function - Late Acceptance Strategy* hyper-heuristic over the 270 ORLib instances in terms of average %-gap, along with the results for *Choice Function - Late Acceptance Strategy* as provided in Table 5.6.

Table 6.4: Comparison between *Choice Function - Late Acceptance Strategy* (CF-LAS) and *Modified Choice Function - Late Acceptance Strategy* (MCF-LAS) on all 270 instances of ORLib in terms of %-gap

Instance Set	CF-LAS	MCF-LAS
OR5x100-0.25	1.16 $_{0.20}$	1.09 $_{0.21}$
OR5x100-0.50	0.53 $_{0.08}$	0.57 $_{0.08}$
OR5x100-0.75	0.40 $_{0.07}$	0.38 $_{0.05}$
OR5x250-0.25	0.42 $_{0.04}$	0.41 $_{0.10}$
OR5x250-0.50	0.20 $_{0.03}$	0.22 $_{0.04}$
OR5x250-0.75	0.13 $_{0.01}$	0.14 $_{0.02}$
OR5x500-0.25	0.19 $_{0.03}$	0.21 $_{0.04}$
OR5x500-0.50	0.10 $_{0.03}$	0.10 $_{0.03}$
OR5x500-0.75	0.06 $_{0.01}$	0.06 $_{0.01}$
OR10x100-0.25	2.00 $_{0.22}$	1.87 $_{0.16}$
OR10x100-0.50	1.02 $_{0.19}$	0.95 $_{0.16}$
OR10x100-0.75	0.58 $_{0.08}$	0.53 $_{0.09}$
OR10x250-0.25	0.83 $_{0.09}$	0.79 $_{0.11}$
OR10x250-0.50	0.39 $_{0.06}$	0.41 $_{0.05}$
OR10x250-0.75	0.23 $_{0.03}$	0.24 $_{0.03}$
OR10x500-0.25	0.40 $_{0.06}$	0.44 $_{0.07}$
OR10x500-0.50	0.18 $_{0.02}$	0.20 $_{0.03}$
OR10x500-0.75	0.12 $_{0.01}$	0.13 $_{0.01}$
OR30x100-0.25	3.45 $_{0.46}$	3.61 $_{0.53}$
OR30x100-0.50	1.56 $_{0.26}$	1.60 $_{0.29}$
OR30x100-0.75	0.92 $_{0.08}$	0.97 $_{0.15}$
OR30x250-0.25	1.55 $_{0.17}$	1.75 $_{0.22}$
OR30x250-0.50	0.71 $_{0.08}$	0.79 $_{0.10}$
OR30x250-0.75	0.39 $_{0.04}$	0.43 $_{0.07}$
OR30x500-0.25	0.92 $_{0.10}$	1.05 $_{0.10}$
OR30x500-0.50	0.39 $_{0.05}$	0.44 $_{0.06}$
OR30x500-0.75	0.23 $_{0.02}$	0.27 $_{0.02}$
Average$_{StdDev}$	0.70 $_{0.09}$	0.73 $_{0.11}$

The results for both hyper-heuristics are very similar over the ORLib instances with

Choice Function - Late Acceptance Strategy obtaining an average %-gap of 0.70 and *Modified Choice Function - Late Acceptance Strategy* a %-gap of 0.73 with little difference in standard deviation. An independent Student's t-test within a 95% confidence interval shows no difference in statistical significance between *Choice Function - Late Acceptance Strategy* and *Modified Choice Function - Late Acceptance Strategy* on this benchmark set. This is in contrast to the results in the previous section where over six different problem domains, significant difference in performance is often observed.

Table 6.5(a) and Table 6.5(b) compare the performance of *Choice Function - Late Acceptance Strategy* and *Modified Choice Function - Late Acceptance Strategy* over the SAC-94 and GK problem instances.

Table 6.5: (a) Success rate of *Choice Function - Late Acceptance Strategy* and *Modified Choice Function - Late Acceptance Strategy* over SAC-94 instances and (b) Performance of *Choice Function - Late Acceptance Strategy* and *Modified Choice Function - Late Acceptance Strategy* on Glover and Kochenberger instances

(a)

Dataset	CF-LAS	MCF-LAS
hp	0.00	0.00
pb	0.67	0.50
pet	0.50	0.50
sento	1.00	1.00
weing	0.63	0.63
weish	1.00	0.90

(b)

Instance	CF-LAS	MCF-LAS
GK01	$0.57_{1.49}$	$0.58_{1.66}$
GK02	$0.81_{3.86}$	$0.78_{3.75}$
GK03	$0.63_{3.10}$	$0.63_{4.30}$
GK04	$0.91_{3.77}$	$0.86_{5.81}$
GK05	$0.45_{3.00}$	$0.44_{5.83}$
GK06	$0.76_{5.02}$	$0.78_{4.04}$
GK07	$0.19_{6.48}$	$0.22_{2.88}$
GK08	$0.33_{5.68}$	$0.31_{7.60}$
GK09	$0.07_{7.47}$	$0.07_{6.85}$
GK10	$0.14_{8.68}$	$0.14_{11.71}$
GK11	$0.13_{12.34}$	$0.14_{11.10}$
Average	$0.45_{0.30}$	$0.45_{0.29}$

Again the two methods are showing virtually identical performance on both of these benchmark sets. *Modified Choice Function - Late Acceptance Strategy* is slightly outperformed in terms of success rate in the pb and weish instances from SAC-94, finding the optimal solution in one and three less instances respectively compared to *Choice Function - Late Acceptance Strategy*. On the GK benchmark set, both methods obtain the same average %-gap over 10 runs of each of the 11 instances.

The results of Chapter 5 suggested that in this problem domain the move acceptance criterion chosen is more important than the heuristic selection mechanism used, echoing the findings of Özcan et al. [173, 174] and Smet et al. [198] amongst others. The combination of the results of this chapter and Chapter 5 suggest that the relationship between performance and heuristic selection and move acceptance is more complex than this and can vary depending on the domain used. It is also possible that generalising the comparison of these components in such a way is not a good idea, and that in fact instance specific dependencies between problems and hyper-heuristic components exist as is the case in Section 6.2.4. A similar notion is discussed by Misir et al. [156], who observed that the move acceptance criteria used can be more important than the heuristic selection mechanism under certain experimental circumstances.

6.4 Concluding Remarks

In this chapter a modified version of the *Choice Function* heuristic selection method is described, managing the parameters which weight the intensification and diversification components of *Choice Function* scores through methods inspired by reinforcement learning. The *Modified Choice Function* aggressively rewards the intensification weighting and heavily punishes the diversification component each time an improvement is made. It is observed that managing these parameters in such a way provides great benefits compared to a classic implementation of the *Choice Function*, when applied to the CHeSC2011 benchmarks. The *Modified Choice Function* combined with *All Moves* move acceptance performs particularly well in the MAX-SAT problem domain, outperforming all of the CHeSC2011 entrants, however the performance over all 6 problem domains is varied. A closer analysis of the two heuristic selection mechanisms reveals that there can be statistically significant domination by one heuristic selection mechanism over the other on different instances within the same domain. The *Modified Choice Function* was then paired with *Late Acceptance Strategy* move acceptance and compared to the best hyper-heuristic of Chapter 5 on three standard benchmark sets for the MKP. Very little variation in performance was observed on this problem domain, where previously move acceptance has been observed to be a more important hyper-heuristic component than heuristic selection method. In this chapter no operators in the HyFlex framework which require more than one input solution such as crossover have been used. The next chapter will introduce such operators into this hyper-heuristic, combining the ideas of this chapter and Chapter 5.

CHAPTER 7

Crossover Control in Cross-domain Optimisation

In the previous chapter a variant of *Choice Function* heuristic selection was presented, showing improved results on average over six problem domains offered by the HyFlex framework. This chapter brings together the crossover management strategies of Chapter 5 and selection hyper-heuristics using the *Modified Choice Function* heuristic selection method proposed in Chapter 6. In Section 7.1, one of the crossover management schemes from Chapter 5 will be used with this hyper-heuristic and applied to the six HyFlex problem domains. The crossover mechanism of the winning entrant of the first Cross-domain Heuristic Search Challenge (CHeSC2011) is introduced in Section 7.2. This crossover mechanism is included in the best performing *Modified Choice Function*-based selection hyper-heuristic in Section 7.2.1. Finally in Section 7.3 the crossover management scheme within the CHeSC2011 winner is replaced by the crossover management strategy from Section 7.1.

7.1 The *Modified Choice Function - All Moves* hyper-heuristic with and without crossover operators in cross-domain optimisation

In Chapter 5 the management of crossover operators in selection hyper-heuristics at two levels of abstraction was investigated. Although against the traditionally accepted definition of hyper-heuristics, which strictly enforces the domain barrier, improved performance was observed by integrating problem domain-specific knowledge. Unfortunately it is not always the case that it is possible to choose the level at which crossover should operate. In fact, if taking the traditional hyper-heuristic model of separating the heuristic and problem search space literally, this is breaking the cardinal rule of crossing the domain barrier. The HyFlex [27, 166] framework introduced in Section 2.5 is one such case where crossover management can only be performed at the hyper-heuristic level. In Chapter 6 the *Modified Choice Function* heuristic selection mechanism was presented and applied to the six problem domains provided by the HyFlex framework. In order to simplify the comparison of heuristic selection mechanisms in Chapter 6 crossover was omitted entirely, however it was observed in Chapter 5 that the use of crossover can be beneficial in some problem domains.

In the CHeSC2011 competition very few of the leading entrants provide a strategy for controlling crossover[1]. In fact of the leading ten entrants, four omit crossover low-level heuristics entirely including two of the top three hyper-heuristics. Two of the top ten do not provide a description of their algorithm at all, so no comment can be made regarding crossover use. A further two include crossover operators but provide no information of how the second solution is chosen. Only two of the top ten hyper-heuristics provide descriptions for managing the second input required by a crossover operator. The first simply uses the current best-of-run solution as the second input solution. The second gives a detailed explanation of a crossover management scheme and was the eventual CHeSC2011 competition winner. It is clear that the management of the second inputs required for crossover is not considered an important part of hyper-heuristic design. Indeed, there are no standard mechanisms defined for controlling crossover in this context. In spite of this, the only method in the top ten which provides a detailed and considered strategy for controlling crossover was the best performing hyper-heuristic overall. It may be no coincidence that the only method to consider anything more than an arbitrary crossover control mechanism won the CHeSC2011 competition.

7.1.1 Crossover management scheme at the hyper-heuristic level

Here crossover is included in a *Modified Choice Function - All Moves* hyper-heuristic, with the second solutions for crossover managed at the hyper-heuristic level as defined in Chapter 5. A memory of elite solutions is maintained from which a second solution, necessary for crossover operators, is selected. An n-ary operator is a low-level heuristic which requires n solutions as input (assuming $n > 1$). In the HyFlex framework the only n-ary operators currently available are crossover operators. Each time a crossover operator is chosen, the first input solution is the current solution. For the second input solution, a random solution is provided from a memory of elite solutions of length

[1]Descriptions of the hyper-heuristics entered into the competition are taken from: `http://www.asap.cs.nott.ac.uk/external/chesc2011/results.html`

m. The memory effectively contains the best *m* solutions found so far. At every *m*-th selection of a crossover operator the elite memory of solutions is not used. Instead, a new solution generated from scratch using the solution initialisation methods provided by the framework is used. If the application of a crossover operator yields an improvement in solution quality compared to the worst solution in the elite memory, the new solution replaces it in the memory, provided that the new solution does not already exist in the memory. This scheme intends to ensure that poor quality solutions early in the search are quickly expunged from the memory, whilst still preserving a certain element of diversity. A generalised pseudocode of this mechanism is shown in Algorithm 9.

Algorithm 9 Crossover input management scheme used in Section 7.1

```
1: Inputs:
2: current solution ($curSoln$)
3: array of solutions in elite memory ($memSoln[m]$)
4: crossover operator selected ($crossOp$)
5: variable to count crossover calls ($calledCount$)
6: if crossover operator is selected then
7:     $calledCount$++
8:     if $calledCount$ mod $m$+1 == 0 then
9:         generate a new solution for crossover, $newCrossSoln$
10:        apply crossover operator - $newSoln \leftarrow crossOp(curSoln, newCrossSoln)$
11:    else
12:        $randIndex \leftarrow$ random Int between 0 and $m$-1
13:        apply crossover operator - $newSoln \leftarrow crossOp(curSoln, memSoln[randIndex])$
14:    end if
15:    if $newSoln$ is better than the worst solution in $memSoln[]$ then
16:        $newSoln$ replaces the worst solution in $memSoln[]$ //if it is not already in
           $memSoln[]$
17:    end if
18: end if
```

The hyper-heuristic level crossover management scheme in Chapter 5 used a memory of solutions as potential input solutions for *n*-ary operators. The length of this memory was relative to the size of instance currently being solved. The notion of instance size is particularly difficult to define when performing cross-domain optimisation. In the case of the CHeSC2011 benchmarks the problems available vary widely in terms of their parameters and characteristics. As a result, the memory length of potential solutions for n-ary operators m is set to 10 for these experiments.

7.1.2 Indirect comparison of *Modified Choice Function - All Moves* with and without crossover over the CHeSC2011 benchmark domains

This section will compare the *Modified Choice Function - All Moves* hyper-heuristic of Chapter 6, which does not use crossover low-level heuristics, with the same hyper-heuristic using the crossover management scheme described in Section 7.1.1. An indirect comparison is performed, ranking each hyper-heuristic against the set of hyper-heuristics submitted to the CHeSC2011 competition over a variety of benchmark domains. As discussed previously, many of the leading entrants of the CHeSC2011 competition do not include crossover low-level heuristics, as no natural mechanism exists to manage the second solution needed for such heuristics. Despite this the winning hyper-heuristic of the CHeSC2011 competition does include a strategy for controlling crossover, suggesting that crossover could be useful in obtaining good quality solutions in a number of domains.

Table 7.1 shows the results of the relative performance of the *Modified Choice Function - All Moves* hyper-heuristics compared to CHeSC2011 competition entrants, using the Formula One scoring system introduced in Section 6.2.3. Table 7.1(a) shows the results of the *Modified Choice Function - All Moves* hyper-heuristic without crossover which was shown to outperform the classic *Choice Function* on these problems in Chapter 6. Table 7.1(b) shows the relative results of *Modified Choice Function - All Moves* including crossover management as described in Section 7.1.1, using the same scoring metrics.

From these tables, it can be seen that including crossover operators clearly gives a marked improvement in performance when compared to the CHeSC2011 entrants. Where the *Modified Choice Function - All Moves* hyper-heuristic without crossover scores 38.85 points, ranking 12th out of 21 hyper-heuristics, the same hyper-heuristic including crossover scores 73.7 points and ranks 6th. The top five hyper-heuristics are unchanged from the original CHeSC2011 competition, with the '*AdapHH*' method of Misir et al. [155] in first place with 177.1 points when ranked against *Modified Choice Function - All Moves* without crossover and 179.35 when compared to *Modified Choice Function - All Moves* with crossover. This is interesting as despite being outperformed by the hyper-heuristic containing crossover on average, the variant which does not include crossover is beaten less often by the best hyper-heuristic overall. This suggests that in at least one problem domain, the *Modified Choice Function - All Moves* hyper-heuristic without crossover is outperforming the *Modified Choice Function - All Moves* hyper-heuristic with crossover. Most of the competition entrants remain in the same relative order

Table 7.1: Results of the median of 31 runs of the *Modified Choice Function - All Moves* hyper-heuristic (a) without crossover and (b) with crossover, compared to CHeSC2011 competitors using Formula One scores over all problem domains

(a)

Rank	Name	Score
1	AdapHH	177.1
2	VNS-TW	131.6
3	ML	127.5
4	PHunter	90.25
5	EPH	88.75
6	NAHH	72.5
7	HAHA	71.85
8	ISEA	68.5
9	KSATS-HH	61.35
10	HAEA	52
11	ACO-HH	39
12	**MCF - AM**	**38.85**
13	GenHive	36.5
14	DynILS	27
15	SA-ILS	22.75
16	XCJ	20.5
17	AVEG-Nep	19.5
18	GISS	16.25
19	SelfSearch	5
20	MCHH-S	3.25
21	Ant-Q	0

(b)

Rank	Name	Score
1	AdapHH	179.35
2	VNS-TW	129.35
3	ML	122
4	PHunter	86.75
5	EPH	84.75
6	**MCF - AM**	**73.7**
7	HAHA	73.6
8	NAHH	70.5
9	ISEA	65.5
10	KSATS-HH	57.2
11	HAEA	49
12	ACO-HH	37
13	GenHive	33.5
14	SA-ILS	22.1
15	DynILS	22
16	XCJ	19.5
17	AVEG-Nep	18.5
18	GISS	16.6
19	SelfSearch	5.5
20	MCHH-S	3.6
21	Ant-Q	0

when *Modified Choice Function - All Moves* with crossover is added for comparison with the exception of *SA-ILS* and *DynILS* which swap places and *NAHH* and *HAHA* which also exchange positions.

Although using the Formula One scoring system as a comparison method gives a good indication of overall performance over all six problem domains, it may be that one method excels in one or more different domains to another. Here the performance in some of the individual problem domains will be presented, again relative to the CHeSC2011 competition entrants. In the previous chapter, Figure 6.2 and Figure 6.3 showed the performance of *Modified Choice Function - All Moves* without crossover which scored 32.85 points in the MAX-SAT problem domain and 6 points in the personnel scheduling domain. This hyper-heuristic scored 0 points in the other 4 domains.

Figure 7.1 shows the number of Formula One points of each of the CHeSC2011 entrants and the *Modified Choice Function - All Moves* hyper-heuristic with crossover in the MAX-SAT problem domain. Here the proposed hyper-heuristic with crossover scores 21.2 points and is the fifth best competitor. Crucially the *Modified Choice Function - All Moves* hyper-heuristic is no longer the highest scoring method, suggesting crossover is in fact detrimental to performance in this domain. Despite the fact that it is no longer the best method in this domain it still offers competitive performance, outperforming 16 of the other 20 hyper-heuristics. The best hyper-heuristic is *AdapHH* [155] which scores 34.1 points.

Figure 7.2 shows the performance of *Modified Choice Function - All Moves* hyper-heuristic with crossover and CHeSC2011 entrants on the personnel scheduling domain using the Formula One scoring system. In this problem domain *Modified Choice Function - All Moves* hyper-heuristic with crossover performs slightly better than *Modified Choice Function - All Moves* hyper-heuristic without crossover, with each method scoring 8.5 points and 6 points respectively. The *Modified Choice Function - All Moves* hyper-heuristic with crossover performs almost as well as the winning CHeSC2011 entrant (*AdapHH*), scoring with only 0.5 points separating these two methods. The best performing hyper-heuristic in personnel scheduling is *VNS-TW* [105] with 37.5 points.

As discussed in Section 6.2.3, the *Modified Choice Function - All Moves* hyper-heuristic without crossover scores 0 points in all other domains. Including crossover leads to points being scored in two further domains, bin packing and the vehicle routing problem.

Figure 7.3 shows the results of the *Modified Choice Function - All Moves* hyper-heuristic with crossover over the bin packing problem instances. This hyper-heuristic ranks third in this domain with 21 points beaten by only two other methods, *ISEA* [131]

Figure 7.1: Number of points scored in the MAX-SAT domain using the Formula One system for each CHeSC2011 competitor and *Modified Choice Function - All Moves* hyper-heuristic with crossover

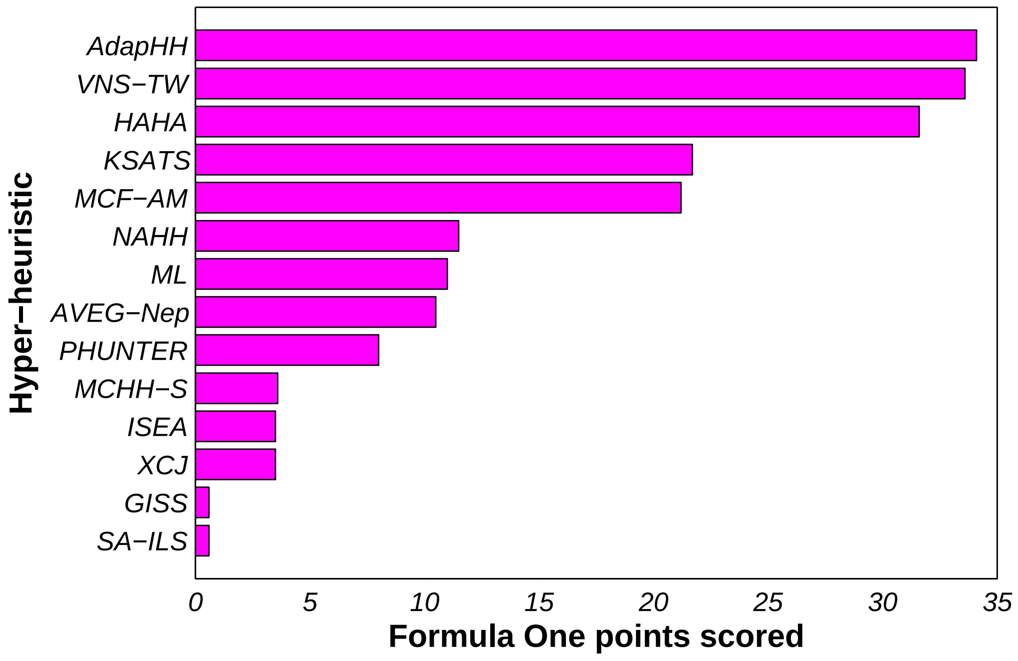


Figure 7.2: Number of points scored in the personnel scheduling domain using the Formula One system for each CHeSC2011 competitor and *Modified Choice Function - All Moves* hyper-heuristic with crossover

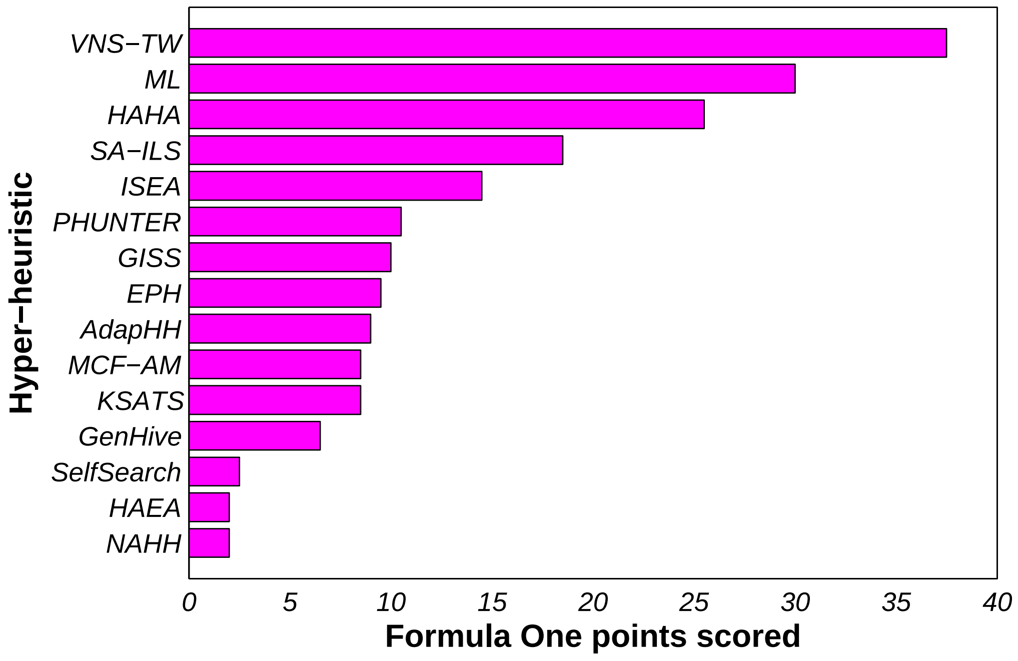
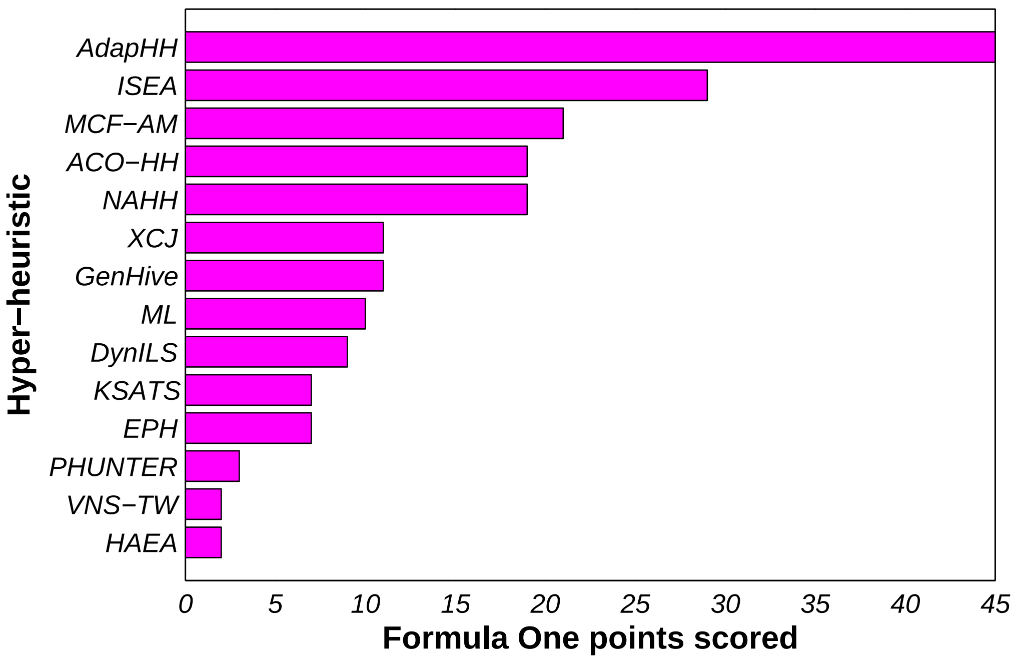


Figure 7.3: Number of points scored in the bin packing domain using the Formula One system for each CHeSC2011 competitor and *Modified Choice Function - All Moves* hyper-heuristic with crossover



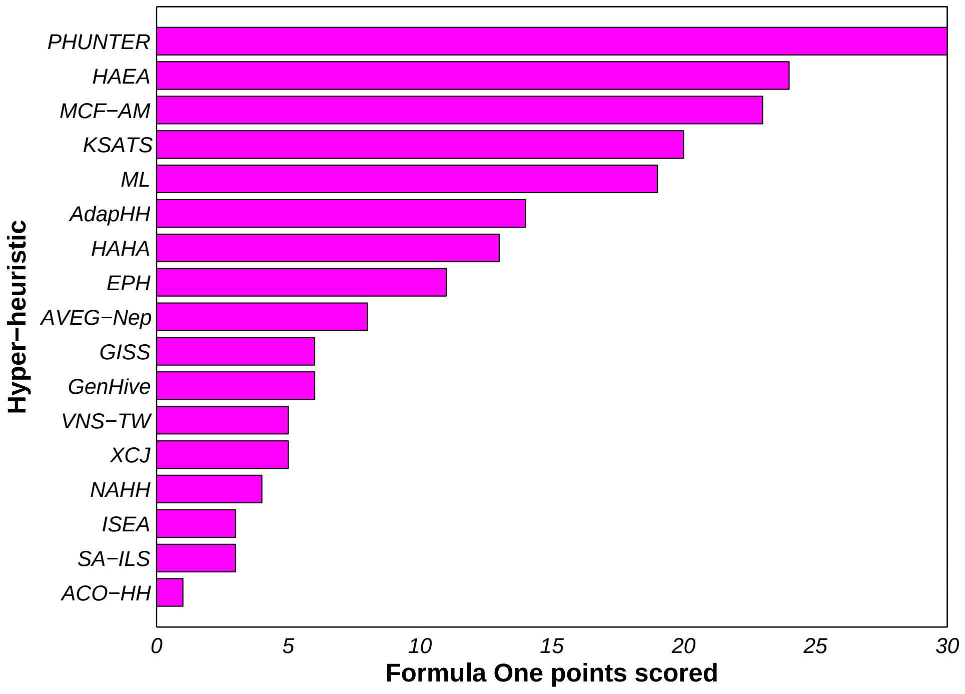
and *AdapHH* which score 29 and 45 points respectively. This is a big improvement on the 0 points scored by the version of this hyper-heuristic without crossover presented in Chapter 6, indicating that crossover greatly improves the solution quality in this domain.

Figure 7.4 presents the results for the hyper-heuristic with crossover over the instances of the vehicle routing problem provided by HyFlex. Again using crossover results in an improvement in performance, obtaining 23 points and ranking third place. Interestingly the other methods in the top three places also utilise crossover operators. The first placed hyper-heuristic is *PHUNTER* [39] with 30 points and second is *HAEA* with 24 points. This suggests that using crossover may be desirable when trying to obtain solutions comparable with state-of-the-art hyper-heuristics in this problem domain.

Despite offering a great improvement in performance in terms of Formula One scores when crossover is added, *Modified Choice Function - All Moves* still scores 0 points in the permutation flow shop and travelling salesman problem domains.

The best performing hyper-heuristics in the permutation flow shop domain are *ML*, *AdapHH* and *VNS-TW*. These are also the top three hyper-heuristics in the competition overall. Both *ML* and *VNS-TW* use an underlying Iterated Local Search [140, 141] framework. Iterated Local Search consists of two phases, 'shaking' and 'local improve-

Figure 7.4: Number of points scored in the vehicle routing problem domain using the Formula One system for each CHeSC2011 competitor and *Modified Choice Function - All Moves* hyper-heuristic with crossover



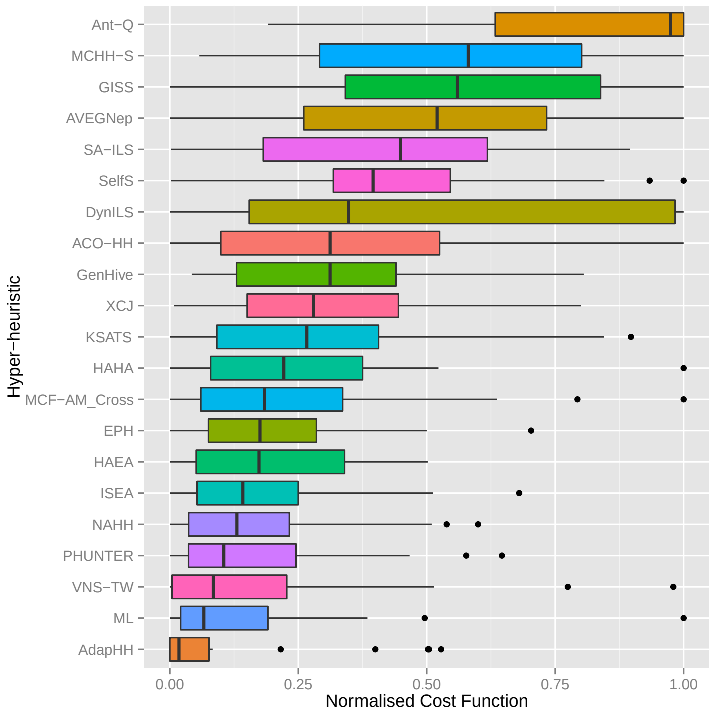
ment'. The shaking phase is applied first and uses perturbation operators to modify the current solution and move the search into a different area of the search space. Following this one or more local search operators are applied in the second phase to move the new solution to a local optimum. In both of these hyper-heuristics, only mutation low-level heuristics are used in the shaking phase, all crossover low-level heuristics are omitted from the set of available heuristics. The F_A and F_C frameworks as defined by Özcan et al. [174] were introduced as hyper-heuristic frameworks in Section 2.3.3. In hyper-heuristic terms these two methods are effectively variants of the F_C framework. Although not strictly tied to the F_C framework, *AdapHH* contains a number of mechanisms which allow it to behave as if it were an F_C hyper-heuristic. It is possible that this hyper-heuristic is behaving in this way in order to be effective in this domain. *Modified Choice Function - All Moves* uses an F_A framework which simply selects and applies a low-level heuristic from the set of available heuristics, with no discrimination between heuristic types. Performance in this domain could be affected by not strictly enforcing local search each time a modification is made in order to reach a local minimum.

In the case of the travelling salesman problem the best three hyper-heuristics are *AdapHH*, *EPH* and *PHUNTER*. Again, these hyper-heuristics are all amongst the top entrants to the CHeSC2011 competition finishing first, fourth and fifth respectively. All of these hyper-heuristics are capable of selecting crossover low-level heuristics, indicating that crossover may be beneficial in this domain. The hyper-heuristics which finish second and third overall, *VNS-TW* and *ML* are fourth and sixth in this problem domain, with another ILS-based hyper-heuristic, *DynILS*, coming 5th. These three hyper-heuristics are all based on the iterative application of a perturbation operator, followed by a local search phase and do not select from the set of crossover low-level heuristics. This suggests that although crossover low-level heuristics are beneficial to the state-of-the-art methods, they are not necessary to obtain above average performance. Despite the fact that the leading entrants are all hyper-heuristics that use crossover, surprisingly *Modified Choice Function - All Moves* with crossover performs badly in this domain. This implies that it may not simply be a case of whether or not to include crossover, and that the best crossover management methods may in fact be domain-specific. It may also be the case that it is not the low-level heuristic set used which determines the quality of solutions found in this domain, but in fact the synergy between other hyper-heuristic components.

Figure 7.5 provides a box whisker plot of the performance of the *Modified Choice Function - All Moves* with crossover compared to the CHeSC2011 entrants, using the normalised objective function of Gaspero and Urli [81] introduced in Section 6.2.3. Again, the 21 hyper-heuristics being compared are sorted by median normalised objective function value with a lower value indicating better performance.

Ranking hyper-heuristics by median normalised objective function value modifies the position of many of the top ten competitors from Table 7.1(b). Effectively this metric measures the distance from the best performing hyper-heuristics in every single instance tested relative to the best and worst performing hyper-heuristics. This could arguably provide a better measure of average performance over all 30 instances than the Formula One scoring system. In any case the best performing hyper-heuristic is still *AdapHH* using this scoring mechanism. *ML* and *VNS-TW* exchange places becoming the second and third best hyper-heuristics respectively. *EPH*, *HAHA* and *Modified Choice Function - All Moves* all drop places, with *Modified Choice Function - All Moves* the eighth best hyper-heuristic when compared this way. Conversely *NAHH* [147], *ISEA* and *HAEA* all rank higher than when compared using the Formula One ranking system. It is likely that those hyper-heuristics which rank in a higher position using the Formula One system perform better in some problem domains than others. The hyper-heuristics placing higher using median normalised objective function value are likely to provide better performance on average over all domains.

Figure 7.5: Box and whisker comparison of 21 CHeSC2011 entrants and *Modified Choice Function - All Moves* with crossover using normalised objective function



7.1.3 Direct comparison of *Modified Choice Function - All Moves* with and without crossover over the HyFlex benchmark domains

The previous section presented an indirect comparison of the *Modified Choice Function - All Moves* hyper-heuristic with crossover omitted and with crossover managed at the hyper-heuristic level using the Formula One scoring system. The comparison was indirect as each method was ranked against a common set of hyper-heuristics which were entered into the CHeSC2011 competition. Whilst this gives a reasonable overview of performance generally and in some specific domains, little can be said of the performance difference in the domains where both methods score 0 points. This section will provide a direct comparison between the objective function values obtained by both hyper-heuristics in 31 runs of each of the 30 competition instances.

Table 7.2 shows the results of an independent Student's t-test within a 95% confidence interval on the objective function values for 31 runs of each instance. For each problem domain, five instances are tested. Each cell of the table provides the number of instances of a particular domain in which there is a variation in performance between the two hyper-heuristics. In this table, $>$ and $\gg$ denote the number of cases that *Modified Choice Function - All Moves* with crossover is outperforming *Modified Choice Function*

Table 7.2: Pairwise comparison of *Modified Choice Function - All Moves* with and without crossover using independent Student's t-test

Problem Domain	$\ll$	$<$	$>$	$\gg$
MAX-SAT	3	2	0	0
Bin Packing	0	0	0	5
Personnel Scheduling	1	1	3	0
Permutation Flow Shop	0	0	3	2
Travelling Salesman Problem	1	0	0	4
Vehicle Routing Problem	1	1	0	3

- *All Moves* without crossover on average or statistically significantly respectively. Conversely, $<$ and $\ll$ denote the number of cases which the *Modified Choice Function - All Moves* without crossover is outperforming *Modified Choice Function - All Moves* with crossover on average or statistically significantly.

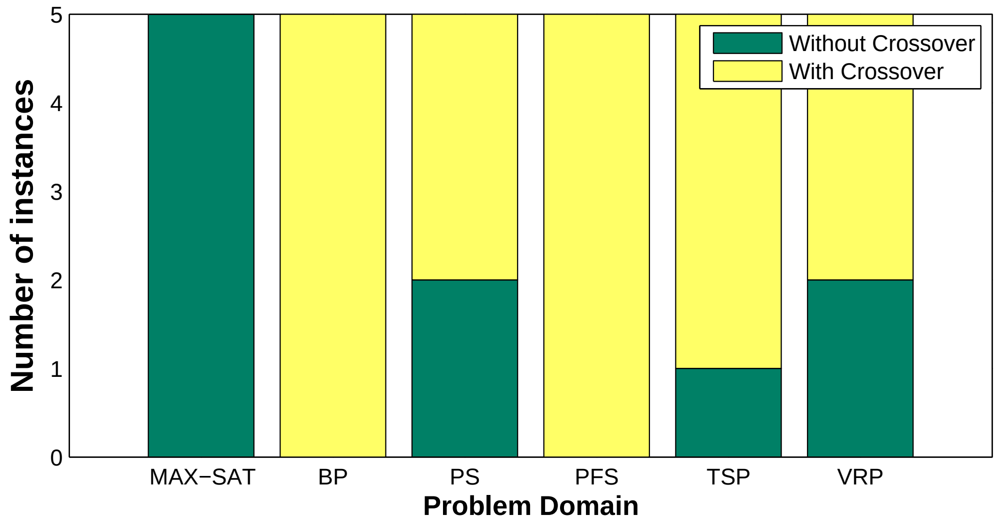
From this table it becomes clear that there is a certain pattern in the performance in some problem domains with respect to whether or not crossover low-level heuristics are used.

In the previous section it was shown that *Modified Choice Function - All Moves* is no longer the best hyper-heuristic in the MAX-SAT domain when crossover low-level heuristics are introduced. A direct comparison between the objective function values shows that the *Modified Choice Function - All Moves* hyper-heuristic without crossover performs better on average in all 5 instances of the competition set, with the difference being statistically significant in 3 instances. Conversely, in bin packing and permutation flow shop, *Modified Choice Function - All Moves* with crossover outperforms *Modified Choice Function - All Moves* without crossover on average in all 5 instances. This difference is statistically significant in all 5 bin packing instances and in 2 of the 5 permutation flow shop instances. In the case of bin packing the difference in performance was noted in the previous section, as there is a clear improvement in relative performance against the CHeSC2011 competitors in this problem domain. For permutation flow shop this difference was less clear in Section 7.1.2 as both methods scored 0 points using the Formula One scoring system.

Differentiating between the performance of each hyper-heuristics in the other three problem domains is more difficult. In personnel scheduling, both methods outperform each other on average in at least one of the instances with the variant not using crossover obtaining statistically significantly better results in one instance. In the case of both the travelling salesman problem and the vehicle routing problem it is the case that either including or omitting crossover low-level heuristics can provide statistically significantly better results depending on the instance in question. This presents a problem when trying to generalise methods as performance does not only vary on a per domain basis but also a per instance basis.

Figure 7.6 shows some of the information of Table 7.2 visually giving the number of instances in which each hyper-heuristic performs best on average. With the exception of MAX-SAT, the problem domains have been abbreviated in this figure as follows: bin packing (BP), personnel scheduling (PS), permutation flow shop (PFS), the travelling salesman problem (TSP) and the vehicle routing problem (VRP).

Figure 7.6: Number of competition instances in which *Modified Choice Function - All Moves* hyper-heuristic with and without crossover perform best on average for each CHeSC2011 problem domain



In terms of the total number of instances in which each hyper-heuristic performed better on average, *Modified Choice Function - All Moves* with crossover is better in 20 cases and *Modified Choice Function - All Moves* without crossover better in 10 cases.

In the case of MAX-SAT, bin packing and permutation flow shop it seems that explicitly including or removing crossover low-level heuristics from the set of available heuristics could potentially lead to improved performance. With the remaining three domains, particularly the travelling salesman problem and the vehicle routing problem, performance can vary significantly depending on the instance being solved so making this decision is less clear cut. Five instances is a small sample from which to provide general comments on the performance of a hyper-heuristic, however it is clear that crossover operators are beneficial in some problem domains and instances and not others.

7.2 Introducing the crossover management scheme of the CHeSC2011 winner into a *Modified Choice Function - All Moves* hyper-heuristic using an F_C framework

The winner of the CHeSC2011 competition, *AdapHH* [155], is currently considered as the state-of-the-art selection hyper-heuristic in cross-domain search. As briefly mentioned in Section 2.3.3, the only method of the top 10 entrants to this competition to provide a strategy was *AdapHH*. Unlike many of the other leading entrants, this hyper-heuristic included crossover operators in the available low-level heuristic set. Although the original description of this algorithm contained no mention of a method to manage the second input solution required by crossover operators, the code of *AdapHH* was later made publicly available with the crossover management scheme defined by Misir [152]:

> '...a population of five solutions including previously explored new best solutions is accommodated. They are applied using the current solution and a randomly selected one from these five solutions. Each time a new best solution is found, a randomly chosen solution from these solutions is replaced by the new solution.'

The following two sections explore the introduction of different high-level hyper-heuristic components, namely crossover input management schemes, into different hyper-heuristics. Section 7.2.1 augments the hyper-heuristics of Section 7.1 to use the F_C framework as defined by Özcan et al. [174]. Following this the crossover management scheme of Misir et al. [155] is introduced into the *Modified Choice Function - All Moves* hyper-heuristic in Section 7.2.2.

7.2.1 The F_C *Modified Choice Function - All Moves* hyper-heuristic with and without crossover

The *Modified Choice Function - All Moves* hyper-heuristics in Section 7.1 operate using an F_A framework. That is, the hyper-heuristics select and apply a low-level heuristic from the full set of available heuristics without discriminating between different heuristic types. This section will modify these hyper-heuristics to operate using an F_C framework. As introduced in Section 2.3.3, Özcan et al. [174] describe four different selection hyper-heuristic frameworks. Using their definition, a selection hyper-heuristic operating within an F_C framework first selects and applies a heuristic from a set of mutational low-level heuristics, before selecting and applying a second low-level heuristic from a set of hill climbers. This is the same framework used for the hyper-heuristics presented in Chapter 5 and shown in Figure 5.2. As discussed in Section 7.1.2, the top three hyper-heuristics in CHeSC2011 are all capable of behaving as F_C selection hyper-heuristics.

Table 7.3 presents the results of *Modified Choice Function - All Moves* with and without crossover, using an F_C selection hyper-heuristic framework over the CHeSC2011 benchmarks. As with the previous experiments, the median of each of 31 ten minute runs are used to compare a hyper-heuristic with the 20 entrants to CHeSC2011 using the Formula One scoring system defined in Section 2.5.3. From Table 7.3(a) and Table 7.3(b) it is clear that the F_C versions of these hyper-heuristics perform slightly worse than their F_A counterparts presented in Section 7.1. Table 7.4 shows the breakdown of points scored in each problem domain, compared to the CHeSC2011 entrants, for each of the F_A and F_C *Modified Choice Function - All Moves* hyper-heuristics with and without crossover .

As seen in Table 7.4, in the case of *Modified Choice Function - All Moves* without crossover, very high scores are obtained in the MAX-SAT problem domain using this hyper-heuristic within an F_C framework. Despite performing well, the F_C hyper-heuristic is not the best method in this domain, unlike the F_A *Modified Choice Function - All Moves* of Chapter 6. The F_C *Modified Choice Function - All Moves* without crossover scores 28.35 points in this domain whereas the F_A version of this hyper-heuristic scored 32.85 points. The performance of these two hyper-heuristics over all six problem domains is very similar in terms of Formula One score. The F_C and F_A *Modified Choice Function - All Moves* hyper-heuristics without crossover score 36.35 and 38.85 points in total against the CHeSC2011 competitors. Both hyper-heuristics score 6 points in the personnel scheduling problem domain. Although the F_C hyper-heuristic scores fewer points in total, it does score 2 points in the vehicle routing problem where previously the F_A hyper-heuristic scored no points. This could be a direct result of the addition of a local improvement phase. The methods which are first, second and third in this domain all operate within frameworks which use local improvement following a perturbative move in the search space.

For *Modified Choice Function - All Moves* with crossover, again the performance of the F_A and F_C variants are similar in terms of total Formula One points scored (73.7 and 73.2 respectively). In the case of MAX-SAT, both variants score exactly the same number of Formula One points (21.2) so the inclusion of a local search phase has little effect in this domain. Both methods are also similar in bin packing and the vehicle routing problem, with the F_C hyper-heuristic slightly outperforming the F_A hyper-heuristic obtaining 24

Table 7.3: Results of the median of 31 runs of F_C *Modified Choice Function - All Moves* hyper-heuristics (a) without crossover and (b) with crossover, compared to CHeSC2011 competitors using Formula One scores over all problem domains

(a)

Rank	Name	Score
1	AdapHH	179.1
2	VNS-TW	131.6
3	ML	127.5
4	PHunter	90.25
5	EPH	89.25
6	HAHA	72.85
7	NAHH	71.5
8	ISEA	68.5
9	KSATS-HH	61.35
10	HAEA	52
11	ACO-HH	39
12	GenHive	36.5
13	**MCF - AM**	**36.35**
14	DynILS	27
15	SA-ILS	22.75
16	XCJ	20.5
17	AVEG-Nep	19.5
18	GISS	16.25
19	SelfSearch	5
20	MCHH-S	3.25
21	Ant-Q	0

(b)

Rank	Name	Score
1	AdapHH	176.85
2	VNS-TW	131.35
3	ML	122
4	PHunter	87.75
5	EPH	84.25
6	HAHA	74.1
7	**MCF - AM**	**73.2**
8	NAHH	69.5
9	ISEA	65.5
10	KSATS-HH	57.7
11	HAEA	49
12	ACO-HH	37
13	GenHive	33.5
14	SA-ILS	23.1
15	DynILS	22
16	XCJ	18.5
17	AVEG-Nep	18.5
18	GISS	16.6
19	SelfSearch	6
20	MCHH-S	3.6
21	Ant-Q	0

Table 7.4: Formula One scores for F_A *Modified Choice Function - All Moves* (a) without crossover, (b) with crossover and F_C *Modified Choice Function - All Moves* (c) without crossover and (d) with crossover in each problem domain compared to CHeSC2011 competitors

(a)

SAT	BP	PS	FS	TSP	VRP	Total
32.85	0	6	0	0	0	38.85

(b)

SAT	BP	PS	FS	TSP	VRP	Total
28.35	0	6	0	0	2	36.35

(c)

SAT	BP	PS	FS	TSP	VRP	Total
21.2	21	8.5	0	0	23	73.7

(d)

SAT	BP	PS	FS	TSP	VRP	Total
21.2	24	2	0	1	25	73.2

and 25 points in each domain respectively, compared to 21 and 23 points gained by the F_A variant. The only notable difference in performance is observed in the personnel scheduling domain where the F_A hyper-heuristic score 8.5 points and the F_C hyper-heuristic scores 2. This is unexpected as in general hyper-heuristics which use a local search phase performed well in this domain in the competition.

7.2.2 The F_C *Modified Choice Function - All Moves* hyper-heuristic with the crossover control scheme of the CHeSC2011 winner

The previous section presented variants of the hyper-heuristics tested in Section 7.1 operating using an F_C framework as defined by Özcan et al. [174]. Here the crossover control scheme of the CHeSC2011 winner will be introduced to manage second input solutions for crossover operators in a *Modified Choice Function - All Moves* hyper-heuristic. Each time a second solution is required by a crossover operator, one is selected at random from a list of 5 solutions. In the case a superior solution is found after applying the crossover low-level heuristic, the new solution replaces one of the 5 solutions in the list at random. Initially the list of 5 solutions is comprised of copies of the starting solution.

Table 7.5 shows the Formula One scores obtained by the *Modified Choice Function - All Moves* hyper-heuristics with the *AdapHH* crossover control scheme, compared to the 20 CHeSC2011 entrants.

Overall the *Modified Choice Function - All Moves* with *AdapHH* crossover scores 34.6 points over the six problem domains, ranking 12th out of 21 when compared to the CHeSC2011 entries. This hyper-heuristic performs very well in the vehicle routing problem, scoring 26 points, second only to *PHUNTER*. In the remaining five problem domains points are scored in four, with 6 points scored in bin packing being the only performance of note.

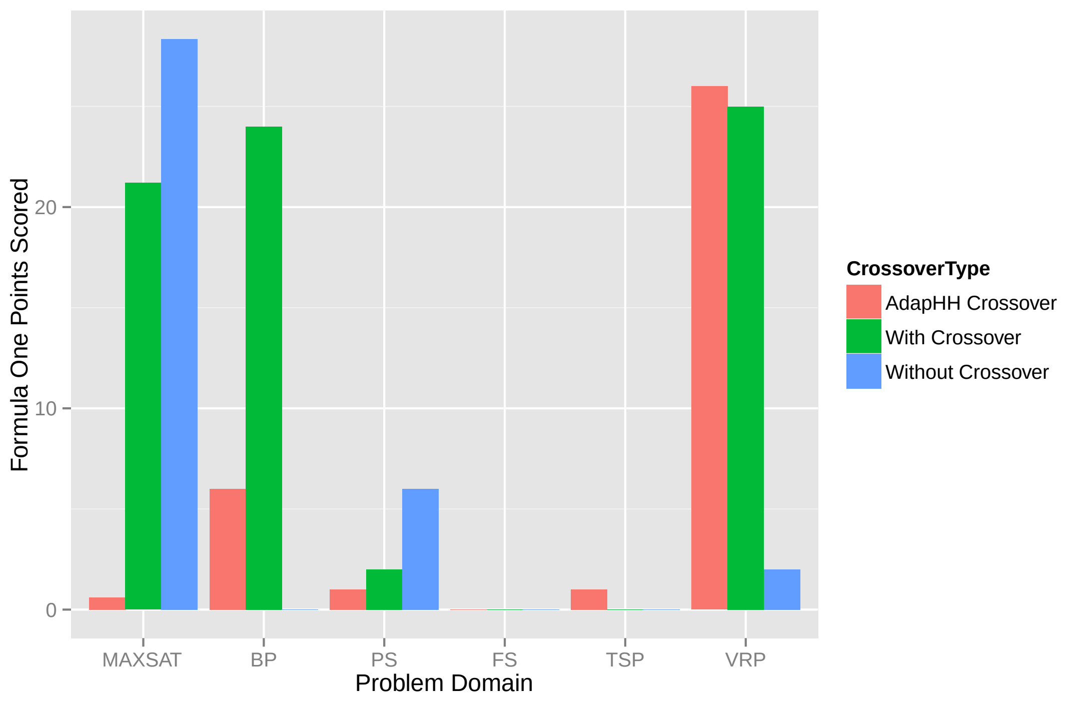
Figure 7.7 shows the number of Formula One points scored by an F_C *Modified Choice Function - All Moves* hyper-heuristic using the crossover control scheme described in Section 7.1.1, the crossover control scheme of *AdapHH* [155] or no crossover at all.

As seen in Figure 7.7, the most striking difference between the hyper-heuristic using the *AdapHH* crossover control scheme and the two *Modified Choice Function - All Moves* hyper-heuristics from Section 7.2.1 (With Crossover and Without Crossover), is the complete deterioration in performance in the MAX-SAT problem domain. Whereas the decision to either include crossover operators or not in the low-level heuristic set did not affect the performance in this domain significantly, the management of the inputs

Table 7.5: Formula One scores for *Modified Choice Function - All Moves* with the crossover control mechanism of the CHeSC2011 winner and CHeSC2011 competitors in each problem domain

Rank	Hyper-heuristic	SAT	BP	PS	FS	TSP	VRP	Total
1	AdapHH	34.75	45	9	37	40.25	14	180
2	VNS-TW	34.25	2	39.5	34	17.25	5	132
3	ML	14.5	11	31	39	13	18	126.5
4	PHunter	10.5	3	11.5	9	26.25	30	90.25
5	EPH	0	10	10.5	21	35.25	11	87.75
6	HAHA	32.75	0	25.5	3.5	0	12	73.75
7	NAHH	14	19	2	22	12	4	73
8	ISEA	6	29	14.5	3.5	12	3	68
9	KSATS-HH	23.85	11	9.5	0	0	20	64.35
10	HAEA	0.5	3	2	10	11	24	50.5
11	ACO-HH	0	20	0	9	8	1	38
12	**MCF - AM**	**0.6**	**6**	**1**	**0**	**1**	**26**	**34.6**
13	GenHive	0	12	6.5	7	3	6	34.5
14	DynILS	0	12	0	0	13	0	25
15	SA-ILS	0.6	0	19.5	0	0	3	23.1
16	XCJ	5.5	12	0	0	0	4	21.5
17	AVEG-Nep	12	0	0	0	0	8	20
18	GISS	0.6	0	10	0	0	6	16.6
19	SelfSearch	0	0	3	0	3	0	6
20	MCHH-S	4.6	0	0	0	0	0	4.6
21	Ant-Q	0	0	0	0	0	0	0

Figure 7.7: Formula One points scored using each crossover control scheme within F_C *Modified Choice Function - All Moves* compared to CHeSC2011 entrants



for such operators can have a great effect on performance. This is an interesting result as it is not necessarily a simple case of whether or not crossover operators are useful in a particular problem class. In the case of the vehicle routing problem it is clear from Figure 7.7 that both of the hyper-heuristics which use crossover outperform the hyper-heuristic which doesn't. This result suggests that crossover is inherently beneficial in the case of the vehicle routing problem irrespective of method of managing the second solutions. Bin packing is another interesting case, where including crossover obviously provides benefits, however the strategy used to manage the input solutions also has a great bearing on performance. The crossover strategy of *AdapHH* scores 6 points in this domain, with the crossover control scheme of Section 7.1.1 scoring 24 points. For the travelling salesman problem and permutation flow shop domains, all three hyper-heuristics perform particularly poorly, scoring a single point between them. It is possible that there is another crossover control scheme which will perform well in these domains, however it is more likely that this is a result of the high-level *heuristic selection method - move acceptance criterion* combination used. The methods that perform particularly well in these two domains are often the hyper-heuristics which perform well overall. Not including crossover operators at all is the best scheme in personnel scheduling however even this strategy does not perform well when compared to the

CHeSC2011 entrants.

7.3 Introducing a different crossover control scheme into the CHeSC2011 winner

Where the previous section introduced the crossover control scheme of the CHeSC2011 winner into an *F_C Modified Choice Function - All Moves hyper-heuristic*, this section will reverse these roles and introduce the crossover control scheme defined in Section 7.1.1 into the CHeSC2011 winner. Following the CHeSC2011 competition the source code for the winning hyper-heuristic (*AdapHH*) was made available[2]. No modifications were made to this source code with the exception of modifying the mechanism to provide second solutions for crossover. Here the results of an independent set of 31 runs of AdapHH over the CHeSC2011 benchmarks are presented along with 31 runs of this hyper-heuristic with the crossover control scheme of Section 7.1.1.

7.3.1 Results

As an indirect comparison, Table 7.6(a) and Table 7.6(b) present the number of Formula One points scored by each *AdapHH* hyper-heuristic when compared with the other 19 competition entrants. These figures are presented visually in Figure 7.8. Please note that in each case the results obtained replace the results of the original CHeSC2011 *AdapHH* hyper-heuristic when calculating the Formula One scores for each instance.

Table 7.6: Formula One scores for (a) an independent run of *AdapHH* and (b) *AdapHH* with modified crossover management scheme in each problem domain compared to CHeSC2011 competitors

(a)

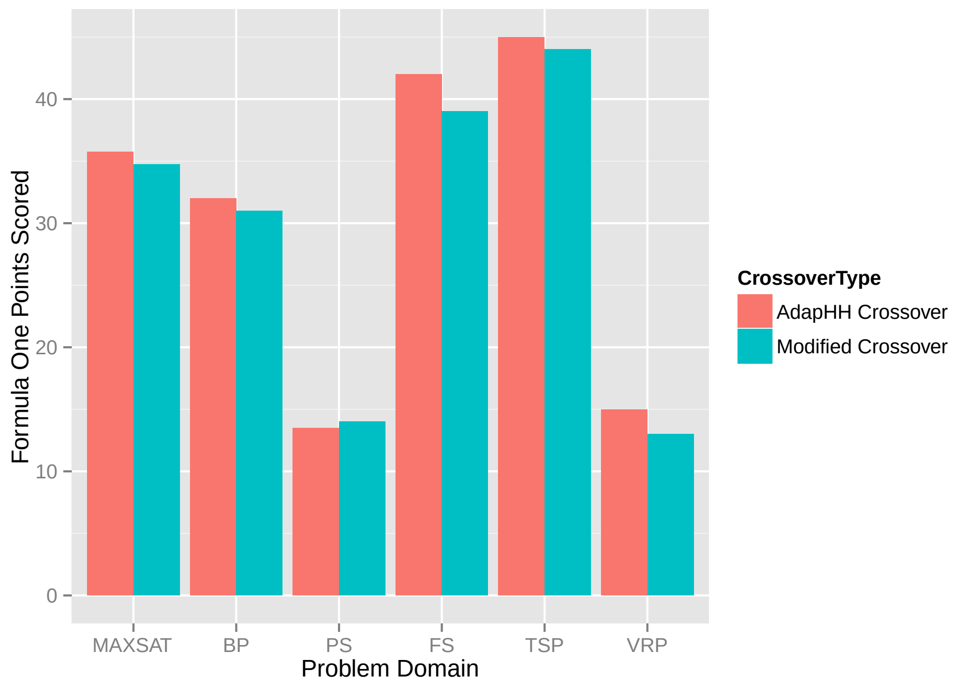
SAT	BP	PS	FS	TSP	VRP	Total
35.75	32	13.5	42	45	15	183.25

(b)

SAT	BP	PS	FS	TSP	VRP	Total
34.75	31	14	39	44	13	175.75

[2]Available at: `http://code.google.com/p/generic-intelligent-hyper-heuristic/downloads/list`

Figure 7.8: Formula One points scored by each crossover control mechanism within *AdapHH* compared to CHeSC2011 competitors



Interestingly the results of 31 independent runs of *AdapHH* yield slightly different results to those achieved in CHeSC2011, scoring 183.25 points overall compared to 181 in the original competition. Comparing the results of Table 7.6 it can be seen that the original crossover management scheme of *AdapHH* is able to yield better results than modifying the crossover management scheme to the method described in Section 7.1.1. The original *AdapHH* outperforms the version with a modified crossover management scheme in all problem domains with the exception of personnel scheduling. There does not appear to be a significant difference in general, with both hyper-heuristics retaining first place when compared to the other 19 competition entrants.

As a direct comparison, Table 7.7 shows the results of an independent Student's t-test within a 95% confidence interval on the objective function values for 31 runs of each instance. Each cell of the table provides the number of instances of a particular domain in which there is a variation in performance between the two hyper-heuristics. In this table $>$ and $\gg$ show the number of cases that the original *AdapHH* is outperforming performing *AdapHH* with a modified crossover control scheme on average or statistically significantly respectively. Conversely, $<$ and $\ll$ denote the number of cases which the *AdapHH* with modified crossover control scheme outperforms the original *AdapHH* on average or statistically significantly.

In terms of average performance, the original *AdapHH* outperforms *AdapHH* with mod-

Table 7.7: Pairwise comparison of *AdapHH* with differing crossover control schemes using independent T-Test

Problem Domain	$\ll$	$<$	$>$	$\gg$
MAX-SAT	0	1	4	0
Bin Packing	0	2	2	1
Personnel Scheduling	0	3	2	0
Permutation Flow Shop	0	3	1	1
Travelling Salesman Problem	0	1	4	0
Vehicle Routing Problem	1	1	2	1

ified crossover management in 18 of the 30 instances. There are very few cases in which the difference is statistically significantly different. An notable case is the vehicle routing problem where it is possible for both *AdapHH* hyper-heuristics to statistically significantly outperform each other in different instances.

7.4 Concluding Remarks

In this chapter, crossover control has been investigated within cross-domain optimisation in a number of ways. Firstly crossover operators have been added to the *Modified Choice Function - All Moves* hyper-heuristic introduced in Chapter 6, managed using a hyper-heuristic level scheme similar to those used in Chapter 5. The inclusion of crossover low-level heuristics results in a large improvement in performance for a *Modified Choice Function - All Moves* hyper-heuristic. It has been observed that crossover seems to provide a greater benefit in some problem domains or instances than others. An interesting question this raises is that if crossover is only beneficial in some circumstances, can methods be designed to recognise when crossover is helpful or not and include it appropriately in a selection hyper-heuristic framework when necessary?

Following this, the hyper-heuristic level crossover control scheme introduced was compared with the crossover management scheme of the CHeSC2011. This was done in two ways, firstly the crossover control scheme defined in Section 7.1 was replaced with the crossover management scheme of *AdapHH* within a *Modified Choice Function - All Moves* hyper-heuristic. Secondly, this crossover control scheme is used within *AdapHH*, replacing the existing crossover control scheme defined for this hyper-heuristic. When comparing different crossover control schemes, it becomes clear that there are some domains or instances for which the choice of crossover control scheme is crucial in determining performance. Unfortunately despite having a large impact in some instances in Section 7.2 within a *Modified Choice Function - All Moves* hyper-heuristic, the choice of crossover management scheme does not make a great deal of difference in the case of *AdapHH*. This provides a challenge when trying to generalise certain hyper-heuristic components, such as the management of solutions for *n*-ary operators, as dependencies clearly exist between different hyper-heuristic components.

It has been noted that although controlling crossover below the domain barrier has shown to be useful in some instances in Chapter 5, depending on the framework used this is not always possible. This raises a broader question regarding the responsibility of crossover management. In the case of the HyFlex framework, this responsibility lies entirely with the implementer of the high-level heuristic search methodology rather than the coder of the low-level problem domain. Although this is essentially a design decision when defining the framework, it would be feasible to extend the HyFlex framework to support domain-level crossover management. This could be done by explicitly allowing the framework to manage potential solutions for heuristics which require more than one solution as input, providing a method which provides access to these solutions to the high-level heuristic search method.

As discussed at the start of this thesis in Section 1.1, Doerr et al. [58, 59] showed that crossover is provably beneficial in at least some classes of practical optimisation problems. If it is indeed the case that the benefit of crossover varies on a per instance basis, this effectively reduces the size of each problem class for which crossover is useful to one, rendering generalisation irrelevant. It could also be the case that in the problem domains where crossover is useful in some cases but not others, there is a subset of instances which share certain features for which the use of crossover leads to gains in performance. This raises a larger question regarding the tuning of hyper-heuristics, particularly in an offline manner. If in advance of a full run, a hyper-heuristic is made aware of instance specific properties (e.g. size, nature of the search landscape), it would be possible to make choices regarding which hyper-heuristic components or parameter settings to use in different cases. A counter argument to this type of tuning is that in effect, as the hyper-heuristic is able to distinguish between problem domains based on these properties and therefore alter behaviour for different domains, the domain barrier is effectively broken.

The same argument can be made regarding offline tuning based on the set of low-level heuristics available, using properties such as heuristic type (i.e. mutation, crossover etc.) or expected runtime. As an example, in the case of the personnel scheduling domain the expected runtime of each low-level heuristic is exceptional in length compared to the other problem domains in HyFlex. Some of the competitors to CHeSC2011 take advantage of this fact, modifying behaviour at execution time based on the observed runtimes of low-level heuristics, often through schemes designed based on previous experience. This is a slippery slope when the intention is to develop general methods which are able to perform well over a large number of domains. If the behaviour of a hyper-heuristic is adjusted in such a way for each available problem domain, potentially it is reduced to a set of if-then clauses, with individual behaviours specified for each problem domain. This reduces the hyper-heuristic design process to a software engineering task, with the intention of 'winning' a competition, rather than focusing on algorithm design for multiple problem domains.

A further complication is the relatively small number of problem domains being considered. In some cases the choice of whether or not to include crossover operators at all will affect performance, however in others it can be affected by the method used to manage the inputs for such operators. In a more general sense, using only six problem domains could be seen as a very small sample. It is accepted that in order to assess the effectiveness of a stochastic method, multiple runs are needed to provide a reasonable picture of average performance. It is also the case that to assess how effective a method is in a given problem domain, a reasonable size sample of problem instances should be tested to average performance. Therefore it could be considered unreasonable to expect any generalisations made using these benchmarks to hold true if there were 50 or 5000 problem domains.

CHAPTER 8

Conclusion

8.1 Context

Hyper-heuristics are a class of high-level search methodologies which function at a higher level of abstraction than traditional techniques, operating on a heuristic search space rather than a search space of solutions. The need for more general methods than those in existence encouraged the development of frameworks such as HyFlex, to support the design of techniques which are able to perform well on a variety of problem domains and instances. This thesis has shown that there are still a number of oversights and unanswered questions within selection hyper-heuristic design.

A number of methods have been developed and applied to a variety of problem domains, with a focus on the use of crossover operators in single-point selection hyper-heuristic search. The management of arguments for low-level heuristics which require more than one solution has been greatly overlooked by previous researchers. A framework in which to define at what level this management occurs at has been proposed, with the responsibility of such decisions given to either the high-level search methodology or the low-level problem implementation. It has been shown that crossover does in fact appear to be beneficial to a search in some cases, however it is possible that this behaviour can be independent of the domain or instance being solved. Developing methods which are genuinely 'general' in the true sense of the word remains an ongoing challenge in hyper-heuristic research. Here, some of the obstacles within developing a general problem solver have been highlighted.

8.2 Summary of Work

8.2.1 Chapter 4

This chapter presented an initial study into the use of hyper-heuristics to solve the multidimensional knapsack problem (MKP). Operating on a set of standard binary low-level heuristics and a set of hyper-heuristic components taken from the literature, twelve hyper-heuristics using a variety of selection methods and acceptance criteria were tested. In this problem domain the choice of acceptance criteria is the determining factor in performance using the given low-level heuristic set.

8.2.2 Chapter 5

Crossover management in selection hyper-heuristics is explored using the MKP as a benchmark in Chapter 5. Crossover management strategies are categorised as operating at one of two levels, either the hyper-heuristic level or the domain level. A new initialisation method for a well-studied problem domain is also introduced. It is observed that in this problem domain, managing crossover arguments at the domain level provides better results than at the hyper-heuristic level.

8.2.3 Chapter 6

In Chapter 6 a tradition selection method, the *Choice Function*, was modified to provide improved performance on cross-domain benchmarks. Over all of the HyFlex benchmark domains tested, the new selection method outperformed the classic *Choice Function* using simple *All Moves* acceptance.

8.2.4 Chapter 7

Chapter 7 explores a number of issues surrounding the use of crossover in selection hyper-heuristics operating over multiple domains. Different schemes for controlling the second arguments for crossover are tested in two different hyper-heuristics. Experiments show that in certain problem domains the use of crossover operators can be beneficial to some hyper-heuristics.

8.3 Extensions and Future Work

A number of possible research directions can be pursued in order to extend the work presented in this thesis.

8.3.1 Further improvement of the *Choice Function* heuristic selection method

Although dynamic, the *Modified Choice Function* heuristic selection mechanism presented in Chapter 6 is fairly inflexible. A particular drawback is the lack of adaptation for hyper-heuristics which operate on low-level heuristic sets with different average application times. For example, the degradation rate in the *Choice Function* weightings might be suitable for a hyper-heuristic which is capable of performing 1,000 low-level heuristics a second, however it may not deal well with a low-level heuristic set which averages 100 low-level heuristic applications in 10 minutes. This variation in number of heuristic applications is evident in the HyFlex framework. In the case of the personnel scheduling problem, the typical number of heuristic applications possible within the 10 minute time limit is in the order of 10^3. In the case of some of the other problem domains, it is not unreasonable to expect the number of low-level heuristic applications within 10 minutes to be in the order of 10^6. It is possible to increase the flexibility of the *Choice Function* through independent management of all parameters including the degradation rate for each measure used in the scoring mechanism.

Another potential area for improvement is to consider setting the parameters of the *Choice Function* as a parameter tuning exercise rather than parameter control as defined by Eiben et al. [66]. There is an existing body of research in automated parameter tuning [158], including methods which use metaheuristics to decide the parameters of a given system in advance of a full run. Potential methods for doing so include Ant Colony Optimisation and Iterated Racing [138]. This has been done previously in the case of the *Choice Function* by Crawford et al. [49], where the weightings were managed with Particle Swarm Optimisation.

8.3.2 Dynamic heuristic selection and move acceptance criteria selection

As the performance of a hyper-heuristic is difficult to predict on a per instance basis, is it possible to extract a set of features of a given problem in advance? If the nature of the search space can be identified before a problem is solved, potentially hyper-heuristic components which are known to perform well on such landscapes can be used. The idea of a (hyper-)hyper-heuristic has been investigated previously by Hyde et al. [109] and Krempser et al. [130]. Potentially an even higher level of abstraction exists where a method selects from a set of hyper-heuristic components to create a hyper-heuristic tailored to solving the current problem instance.

8.3.3 Closer integration of selection hyper-heuristic and generation hyper-heuristic paradigms

Currently the two main branches of hyper-heuristic research, heuristic selection and heuristic generation research are very disparate. An interesting direction would be to try to integrate these branches by searching over a set of low-level heuristics which are adapting during a search. If the set of low-level heuristics used by a selection hyper-heuristic is inadequate at a given time, a heuristic generation hyper-heuristic could be used to modify the set of low-level heuristics available.

8.3.4 Automation of crossover application strategies and experimentation over a wider variety of conditions using the HyFlex framework

In Chapter 7 it is observed that including or omitting crossover can yield statistically significant differences in performance, not only between different problem domains but also between different instances within a problem domain. These observations are made after a period of 10 minutes execution time as defined by the CHeSC2011 competition benchmarks. An interesting direction would be to analyse whether this behaviour can be seen at an earlier stage. If it can be ascertained that crossover is not helpful to the search process early on, a decision can be taken whether to proceed with using operators in this category. It would be possible to integrate such a decision mechanism in any of the *Modified Choice Function - All Moves* variants presented in Chapter 7, or indeed into the CHeSC2011 winner *AdapHH*. Such an approach would fall under the category of online learning as a decision is taken during the search process.

Alternatively this decision could be made in an offline manner in advance of a run. Assuming that some a-priori information exists (in the case of the MKP the number of variables, dimensions and tightness ratio are available), can a decision to include or omit crossover be taken using these features? This is similar to the ideas used in Algorithm Portfolios [91], where component algorithms are combined based on 'expected computational cost' and overall risk.

Much of the existing work using the HyFlex benchmarks has been constrained by a fairly arbitrary measure of time as defined by the CHeSC2011 competition rules. There is no evidence to suggest that a period of 10 minutes is an appropriate amount of time to ascertain the benefit of a given crossover management strategy. Any further work which seeks to adaptively decide how to manage crossover needs to be verified by performing runs of different lengths to ensure that the correct decision is made.

8.4 Final Remarks

This thesis has explored a number of issues within hyper-heuristics with a focus on the use of crossover low-level heuristics within a single-point search framework. Although hyper-heuristic research is still relatively young compared to some well-known existing metaheuristic approaches, many gains have been seen in real-world problems as a result of using such methods. Despite this, there is still a long way to go in terms of realising the original goals of hyper-heuristic systems. In order for hyper-heuristics to operate at a truly higher-level of abstraction than traditional search techniques, a greater emphasis on generality must be explored. Here it is observed that not only is performance both domain and instance independent for a given hyper-heuristic, the performance of different hyper-heuristic components is also often independent of the problem being solved. Many hyper-heuristic methods, including some of those included in this thesis, perform offline tuning in order to set the parameters for the set of instances it expects to encounter. In order for hyper-heuristics to genuinely reach a high level of generality, it could be argued that it is necessary for all parameters to be controlled in an adaptive online manner. Hyper-heuristics of this nature would be closer to the idea of an 'oracle' general problem solver, able to effectively manage any set of problems or experimental conditions it encounters.

APPENDIX A

Categorisations of selection hyper-heuristics discussed in the literature review

Table A.1: Categorisation of recent selection hyper-heuristics based on selection method used

Heuristic Selection Method	References
Choice Function	[12, 15, 17, 35, 124, 126, 198]
Greedy	[12, 35, 77, 124, 126]
Probability-based Selection	[8, 17, 43, 77, 130, 150, 156, 151, 189]
Random Descent	[12, 126, 128]
Permutation-based Selection	[12]
Reinforcement Learning	[17, 35, 124, 126, 125, 128, 149, 150]
Simple Random	[12, 15, 35, 43, 56, 77, 124, 126, 130, 128, 150, 153, 156, 151, 198, 211]
Tabu Search	[15]

Table A.2: Categorisation of recent selection hyper-heuristics based on acceptance criterion used

Acceptance Criterion	References
All Moves	[12, 17, 77, 126, 128, 156, 151]
Threshold-based Acceptance	[12, 15, 56, 124, 126, 150, 153, 156, 151, 198, 211]
Improving and Equal	[12, 15, 56, 126, 125, 128, 153, 156, 151, 198, 211]
Late Acceptance	[56, 153, 156, 151]
Only Improving	[12, 15, 126, 130, 156, 151]
Simulated Annealing	[8, 15, 17, 35, 56, 126, 153, 156, 151, 189, 198, 211]

Table A.3: Categorisation of recent selection hyper-heuristics based on problem domain solved

Problem Domain	References
Bin Packing	[8]
DNA Sequence Prediction	[17]
DTLZ Benchmark	[150]
Dynamic Environments	[124, 126, 125, 128]
p-median Problem	[188]
Satisfiability	[149]
Scheduling Problems	[12, 77, 153, 156, 189, 198]
Timetabling	[8, 35, 56, 116, 117]
Vehicle Routing Problems	[153, 189]
Web Document Clustering	[43]

Table A.4: Categorisation of recent selection hyper-heuristics applied to the HyFlex benchmarks based on selection method used

Heuristic Selection Method	References
Choice Function	[32, 116]
Greedy	[41, 116, 117]
Probability-based Selection	[133, 150, 154]
Random Descent	[171]
Permutation-based Selection	[41, 122, 131]
Reinforcement Learning	[32, 81, 116, 150, 209]
Simple Random	[32, 41, 112, 116, 150]

Table A.5: Categorisation of recent selection hyper-heuristics applied to the HyFlex benchmarks based on acceptance criterion used

Acceptance Criterion	References
All Moves	[41, 81, 131, 164, 209]
Threshold-based Acceptance	[122, 150, 154]
Improving and Equal	[105, 133, 154]
Late Acceptance	[112]
Only Improving	[32, 164]
Simulated Annealing	[116, 117, 154, 164]

References

[1] Micah Adler, Phillip B. Gibbons, and Yossi Matias. Scheduling space-sharing for internet advertising. *Journal of Scheduling*, 5(2):103–119, 2002.

[2] Yalçıin Akçay, Haijun Li, and Susan H Xu. Greedy algorithm for the general multidimensional knapsack problem. *Annals of Operations Research*, 150(1):17–29, 2007.

[3] Enrico Angelelli, Renata Mansini, and M. Grazia Speranza. Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers & Operations Research*, 37(11):2017–2026, 2010.

[4] Josep Argelich, Chu-Min Li, Felip Manya, and Jordi Planes. Maxsat evaluation 2009 benchmark data sets. Online, 2009. URL `http://www.maxsat.udl.cat/`.

[5] Shahriar Asta, Ender Özcan, and Andrew J. Parkes. Dimension reduction in the search for online bin packing policies. In Christian Blum and Enrique Alba, editors, *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO 2013)*, pages 65–66, Amsterdam, The Netherlands, 2013. ACM.

[6] Shahriar Asta, Ender Özcan, Andrew J. Parkes, and A. Şima Etaner-Uyar. Generalizing hyper-heuristics via apprenticeship learning. In Martin Middendorf and Christian Blum, editors, *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2013)*, volume 7832 of *LNCS*, pages 169–178, Vienna, Austria, 2013. Springer.

[7] Mohamed Bader-El-Den and Riccardo Poli. Generating sat local-search heuristics using a gp hyper-heuristic framework. In Nicolas Monmarché, El-Ghazali Talbi, Pierre Collet, Marc Schoenauer, and Evelyne Lutton, editors, *Proceedings of the International Conference on Artificial Evolution (EA 2007)*, volume 4926 of *LNCS*, pages 37–49, Tours, France, 2008. Springer.

[8] Ruibin Bai, Jacek Blazewicz, Edmund K. Burke, Graham Kendall, and Barry Mc-

Collum. A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR: A Quarterly Journal of Operations Research*, 10(1):43–66, 2012.

[9] Egon Balas and Eitan Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28(5):1130–1154, 1980.

[10] Roberto Battiti. *Modern heuristic search methods*, chapter Reactive search: Toward self-tuning heuristics, pages 61–83. Wiley, 1996.

[11] Roberto Battiti, Mauro Brunato, and Franco Mascia, editors. *Reactive search and intelligent optimization*. Springer, 2008.

[12] Argun Berberoglu and A. Sima Uyar. Experimental comparison of selection hyper-heuristics for the short-term electrical power generation scheduling problem. In Cecilia Di Chio, Anthony Brabazon, Gianni A. Di Caro, Rolf Drechsler, Muddassar Farooq, Jörn Grahl, Gary Greenfield, Christian Prins, Juan Romero, Giovanni Squillero, Ernesto Tarantino, Andrea Tettamanzi, Neil Urquhart, and A. Sima Etaner-Uyar, editors, *Proceedings of the International Conference on the Applications of Evolutionary Computation (EvoApplications 2011)*, volume 6625 of *LNCS*, pages 444–453, Torino, Italy, 2011. Springer.

[13] Leonardo Bezerra, Manuel López-Ibáñez, and Thomas Stützle. Automatic generation of moaco algorithms for the biobjective bidimensional knapsack problem. In Marco Dorigo, Mauro Birattari, Christian Blum, Anders Lyhne Christensen, Andries Petrus Engelbrecht, Roderich Groß, and Thomas Stützle, editors, *Proceedings of Swarm Intelligence - 8th International Conference (ANTS 2012)*, volume 7461 of *LNCS*, pages 37–48, Brussels, Belgium, 2012. Springer.

[14] Burak Bilgin, Ender Özcan, and Emin Erkan Korkmaz. An experimental study on hyper-heuristics and exam timetabling. In Edmund K. Burke and Hana Rudová, editors, *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, volume 3867 of *LNCS*, pages 394–412, Brno, Czech Republic, 2006. Springer.

[15] Burak Bilgin, Peter Demeester, Mustafa Misir, Wim Vancroonenburg, and Greet Vanden Berghe. One hyperheuristic approach to two timetabling problems in health care. *Journal of Heuristics*, 18(3):401–434, 2012.

[16] Duncan Black, Robert Albert Newing, Iain McLean, Alistair McMillan, and Burt L. Monroe. *The theory of committees and elections*. Cambridge: University Press, 1958.

[17] Jacek Blazewicz, Edmund K. Burke, Graham Kendall, Wojciech Mruczkiewicz, Ceyda Oguz, and Aleksandra Swiercz. A hyper-heuristic approach to sequencing by hybridization of dna sequences. *Annals of Operations Research*, 207(1):27–41, 2013.

[18] Sylvain Boussier, Michel Vasquez, Yannick Vimont, Saïd Hanafi, and Philippe Michelon. A multi-level search strategy for the 0-1 multidimensional knapsack problem. *Discrete Applied Mathematics*, 158(2):97–109, 2010.

[19] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.

[20] Edmund K. Burke and Yuri Bykov. A late acceptance strategy in hill-climbing for exam timetabling problems. In *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, page Extended Abstract, Montreal, Canada, 2008.

[21] Edmund K. Burke, Emma Hart, Graham Kendall, Jim Newall, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, chapter 16, pages 457–474. Kluwer Academic Publishers, 2003.

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

[23] Edmund K. Burke, Dario Landa-Silva, and Eric Soubeiga. *Meta-heuristics: Progress as Real Problem Solvers*, chapter Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling, pages 129–158. Springer, 2005.

[24] Edmund K. Burke, Matthew Hyde, and Graham Kendall. Evolving bin packing heuristics with genetic programming. In Thomas Philip Runarsson, Hans-Georg Beyer, Edmund K. Burke, Juan J. Merelo Guervós, L. Darrell Whitley, and Xin Yao, editors, *Proceedings of Parallel Problem Solving from Nature (PPSN 2006)*, volume 4193 of *LNCS*, pages 860–869, Reykjavik, Iceland, 2006. Springer.

[25] Edmund K. Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, 2007.

[26] Edmund K. Burke, John Woodward, Matthew Hyde, and Graham Kendall. Automatic heuristic generation with genetic programming: Evolving a jack-of-alltrades or a master of one. In Hod Lipson, editor, *Proceedings of the Genetic*

and Evolutionary Computation Conference (GECCO 2007), pages 1559–1565, London, UK, 2007. ACM.

[27] Edmund K. Burke, Tim Curtois, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Sanja Petrovic, and José Antonio Vázquez-Rodríguez. Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In *Proceedings of the Multidisciplinary International conference on Scheduling: Theory and Applications (MISTA 2009)*, pages 790–797, Dublin, Ireland, 2009.

[28] Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. *Computational Intelligence: Collaboration, Fusion and Emergence*, chapter Exploring Hyper-heuristic Methodologies with Genetic Programming, pages 177–201. Springer, 2009.

[29] Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John Woodward. *Handbook of Metaheuristics 2nd ed.*, chapter A Classification of Hyper-heuristic Approaches, pages 449–468. Springer, 2010.

[30] Edmund K. Burke, Matthew Hyde, Graham Kendall, and John Woodward. A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942–958, 2010.

[31] Edmund K. Burke, Michel. Gendreau, Matthew. Hyde, Graham. Kendall, Barry. McCollum, Gabriela. Ochoa, Andrew J. Parkes, and Sanja. Petrovic. The cross-domain heuristic search challenge - an international research competition. In Carlos A. Coello Coello, editor, *Proceedings of Learning and Intelligent Optimization (LION 2011)*, volume 6683 of *LNCS*, pages 631–634, Rome, Italy, 2011. Springer.

[32] Edmund K. Burke, Michel Gendreau, Gabriela Ochoa, and James D. Walker. Adaptive iterated local search for cross-domain optimisation. In Natalio Krasnogor and Pier Luca Lanzi, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*, pages 1987–1994, Dublin, Ireland, 2011. ACM.

[33] Edmund K. Burke, Matthew Hyde, and Graham Kendall. Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation*, 16(3): 406–417, 2012.

[34] Edmund K. Burke, Matthew Hyde, Graham Kendall, and John Woodward. Automating the packing heuristic design process with genetic programming. *Evolutionary Computation (MIT Press)*, 20(1):63–89, 2012.

[35] Edmund K. Burke, Graham Kendall, Mustafa Misir, and Ender Özcan. Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 196(1):73–90, 2012.

[36] Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[37] Konstantin Chakhlevitch and Peter Cowling. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In Günther R. Raidl and Jens Gottlieb, editors, *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)*, volume 3448 of *LNCS*, pages 23–33, Lausanne, Switzerland, 2005. Springer.

[38] Konstantin Chakhlevitch and Peter Cowling. Hyperheuristics: Recent developments. In Carlos Cotta, Marc Sevaux, and Kenneth Sörensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. Springer, 2008.

[39] Ching-Yuen Chan, Fan Xue, W.H. Ip, and C.F. Cheung. A hyper-heuristic inspired by pearl hunting. In Youssef Hamadi and Marc Schoenauer, editors, *Proceedings of Learning and Intelligent Optimization (LION 2012)*, volume 7219 of *LNCS*, pages 349–353, Paris, France, 2012. Springer.

[40] Paul C. Chu and John E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998. ISSN 1381-1231.

[41] Tomasz Cichowicz, Maciej Drozdowski, Michal Frankiewicz, Grzegorz Pawlak, Filip Rytwinski, and Jacek Wasilewski. Five phase and genetic hive hyper-heuristics for the cross-domain search. In Youssef Hamadi and Marc Schoenauer, editors, *Proceedings of Learning and Intelligent Optimization (LION 2012)*, volume 7219 of *LNCS*, pages 354–359, Paris, France, 2012. Springer.

[42] Robert Cleary and Michael O'Neill. An attribute grammar decoder for the 0/1 multiconstrained knapsack problem. In Günther R. Raidl and Jens Gottlieb, editors, *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)*, volume 3448 of *LNCS*, pages 34–45, Lausanne, Switzerland, 2005. Springer.

[43] Carlos Cobos, Martha Mendoza, and Elizabeth Leon. A hyper-heuristic approach to design and tuning heuristic methods for web document clustering. In *Proceed-*

ings of the IEEE Congress on Evolutionary Computation (CEC 2011), pages 1350–1358, New Orleans, LA, USA, 2011. IEEE Press.

[44] Jean-François Cordeau, Michel Gendreau, Gilbert Laporte, Jean-Yves Potvin, and François Semet. A guide to vehicle routing heuristics. *The Journal of the Operational Research Society*, 53(5):512–522, 2002.

[45] Elon Santos Correa, Maria Teresinha A. Steiner, Alex A. Freitas, and Celso Carnieri. A genetic algorithm for the p-median problem. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 1268–1275, San Francisco, CA, USA, 2001. Morgan Kaufmann.

[46] Carlos Cotta and José Troya. *Artificial Neural Nets and Genetic Algorithms*, chapter A Hybrid Genetic Algorithm for the 0-1 Multiple Knapsack Problem, pages 250–254. Springer, 1998.

[47] Peter Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In Edmund K. Burke and Wilhelm Erben, editors, *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2000)*, volume 2079 of *LNCS*, pages 176–190, Konstanz, Germany, 2001. Springer.

[48] Peter Cowling, Graham Kendall, and Eric Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the Metahuristics International Conference (MIC 2001)*, pages 127–131, Porto, Portugal, 2001.

[49] Broderick Crawford, Ricardo Soto, Eric Monfroy, Wenceslao Palma, Carlos Castro, and Fernando Paredese. Parameter tuning of a choice-function based hyperheuristic using particle swarm optimization. *Expert Systems with Applications*, 40 (5):1690–1695, 2013.

[50] CRIL. Sat competition 2007 benchmark data sets. Online, 2007. URL `http://www.cril.univ-artois.fr/SAT07/`.

[51] CRIL. Sat competition 2009 benchmark data sets. Online, 2009. URL `http://www.cril.univ-artois.fr/SAT09/`.

[52] Federico Della Croce and Andrea Grosso. Improved core problem based heuristics for the 0-1 multi-dimensional knapsack problem. *Computers & Operations Research*, 39(1):27–31, 2012.

[53] Tim Curtois. Staff rostering benchmark data sets. Online, 2009. URL `http:///www.cs.nott.ac.uk/~tec/NRP/`.

[54] Lawrence Davis. Bit-climbing, representational bias, and test suite design. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the International Conference on Genetic Algorithms (ICGA 1991)*, pages 18–23, San Diego, CA, USA, 1991. Morgan Kaufmann.

[55] Richard Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, UK, 2006.

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

[57] Jörg Denzinger, Matthias Fuchs, and Marc Fuchs. High performance atp systems by combining several ai methods. Technical report, SEKI-Report SR-96-09, University of Kaiserslautern, 1996.

[58] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. In Conor Ryan and Maarten Keijzer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 539–546, Atlanta, Georgia, USA, 2008. ACM.

[59] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 436:71–86, 2012.

[60] Jack J. Dongarra. Performance of various computers using standard linear equations software. Online, 2013. URL `http://www.netlib.org/benchmark/performance.pdf`. retrieved 22/05/2013.

[61] John H. Drake, Matthew Hyde, Khaled Ibrahim, and Ender Özcan. A genetic programming hyper-heuristic for the multidimensional knapsack problem. In *Proceedings of the 11th IEEE International Conference on Cybernetic Intelligent Systems (CIS 2012)*, pages 76–80, Limerick, Ireland, 2012. IEEE Press.

[62] John H. Drake, Ender Özcan, and Edmund K. Burke. An improved choice function heuristic selection for cross domain heuristic search. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Proceedings of Parallel Problem Solving from Nature (PPSN 2012), Part II*, volume 7492 of *LNCS*, pages 307–316, Taormina, Italy, 2012. Springer.

[63] John H. Drake, Nikolaos Kililis, and Ender Özcan. Generation of vns components with grammatical evolution for vehicle routing. In Krzysztof Krawiec, Alberto Moraglio, Ting Hu, A. Sima Etaner-Uyar, and Bin Hu, editors, *Genetic Programming - 16th European Conference (EuroGP 2013)*, volume 7831 of *LNCS*, pages 25–36, Vienna, Austria, 2013. Springer.

[64] A. Drexl. A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing*, 40(1):1–8, 1988.

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

[66] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[67] ESICUP. European special interest group on cutting and packing benchmark data sets. Online, 2011. URL `http://paginas.fe.up.pt/~esicup/`.

[68] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Extreme value based adaptive operator selection. In Günter Rudolph, Thomas Jansen, Simon M. Lucas, Carlo Poloni, and Nicola Beume, editors, *Proceedings of Parallel Problem Solving from Nature (PPSN 2008)*, volume 5199 of *LNCS*, pages 175–184, Dortmund, Germany, 2008. Springer.

[69] Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Toward comparison-based adaptive operator selection. In Martin Pelikan and Jürgen Branke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, pages 767–774, Portland, Oregon, USA, 2010. ACM.

[70] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In *Factory Scheduling Conference*, Carnegie Institute of Technology, 1961.

[71] Krzysztof Fleszar and Khalil S. Hindi. Fast, effective heuristics for the 0-1 multidimensional knapsack problem. *Computers & Operations Research*, 36(5):1602–1607, 2009. ISSN 0305-0548.

[72] Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building block hypothesis. In L. Darrell Whitley, editor, *Proceedings of Foundations of Genetic Algorithms (FOGA 1992)*, pages 109–126, Vail, Colorado, USA, 1992. Morgan Kaufmann.

[73] Arnaud Freville and Gérard Plateau. An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. *Discrete Applied Mathematics*, 49(1-3):189–212, 1994.

[74] Alex S. Fukunaga. Automated discovery of composite sat variable-selection heuristics. In Rina Dechter and Richard S. Sutton, editors, *Proceedings of Eighteenth National Conference on Artificial Intelligence*, pages 641–648, Edmonton, Alberta, Canada, 2002. MIT Press.

[75] Alex S. Fukunaga. Evolving local search heuristics for sat using genetic programming. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), Part II*, volume 3103 of *LNCS*, pages 483–494, Seattle, WA, USA, 2004. Springer.

[76] Alex S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation (MIT Press)*, 16(1):31–61, 2008.

[77] Alberto García-Villoria, Said Salhi, Albert Corominas, and Rafael Pastor. Hyper-heuristic approaches for the response time variability problem. *European Journal of Operational Research*, 211(1):160–169, 2011.

[78] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[79] Pablo Garrido and Carlos Castro. Stable solving of cvrps using hyperheuristics. In Franz Rothlauf, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2009)*, pages 255–262, Québec, Canada, 2009. ACM.

[80] Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetabling. In Edmund K. Burke and Wilhelm Erben, editors, *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, volume 2079 of *LNCS*, pages 104–117, Konstanz, Germany, 2000. Springer.

[81] Luca Di Gaspero and Tommaso Urli. Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In Youssef Hamadi and Marc Schoenauer, editors, *Proceedings of Learning and Intelligent Optimization (LION 2012)*, volume 7219 of *LNCS*, pages 384–389, Paris, France, 2012. Springer.

[82] C. D. Geiger, R. Uzsoy, and H. Aytug. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling*, 9(1): 7–34, 2006.

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

[84] Jonathon Gibbs, Graham Kendall, and Ender Özcan. Scheduling english football fixtures over the holiday period using hyper-heuristics. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph, editors, *Proceedings of Parallel Problem Solving from Nature (PPSN 2011)*, volume 6238 of *LNCS*, pages 496–505, Kraków, Poland, 2011. Springer.

[85] John C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41(2):148–177, 1979.

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

[87] Fred Glover. Tabu search - part 1. *ORSA Journal on Computing*, 1(2):190–206, 1989.

[88] Fred Glover and Gary Kochenberger. Benchmarks for "the multiple knapsack problem". Online, n.d. URL `http://hces.bus.olemiss.edu/tools.html`. retrieved 03/02/2012.

[89] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[90] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Boston, MA, USA, 1989. ISBN 0201157675.

[91] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126 (1-2):43–62, 2001.

[92] Jens Gottlieb. On the effectivity of evolutionary algorithms for the multidimensional knapsack problem. In Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, Edmund M. A. Ronald, and Marc Schoenauer, editors, *Proceedings of Artificial Evolution (AE 1999)*, volume 1829 of *LNCS*, pages 23–37, Dunkerque, France, 2000. Springer.

[93] Martin T. Hagan, Howard B. Demuth, and Mark H. Beale. *Neural network design*. PWS Publishing, 1996.

[94] Saïd Hanafi and Christophe Wilbaut. Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Annals of Operations Research*, 183(1): 125–142, 2011.

[95] Saïd Hanafi, Jasmina Lazic, Nenad Mladenovic, Christophe Wilbaut, and Igor Crevits. New hybrid matheuristics for solving the multidimensional knapsack problem. In Maria J. Blesa, Christian Blum, Günther R. Raidl, Andrea Roli, and Michael Sampels, editors, *Proceedings of the International Conference on Hybrid Metaheuristics (HM 2010)*, volume 6373 of *LNCS*, pages 118–132, Vienna, Austria, 2010. Springer.

[96] Pierre Hansen, Nenad Mladenović, and Dionisio Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.

[97] Ami Hauptman, Achiya Elyasaf, and Moshe Sipper. Evolving hyper heuristic-based solvers for rush hour and freecell. In Ariel Felner and Nathan R. Sturtevant, editors, *Proceedings of the Third Annual Symposium on Combinatorial Search (SOCS 2010)*, pages 149–150, Atlanta, Georgia, USA, 2010. AAAI Press.

[98] Jun He, Feidun He, and Hongbin Dong. Pure strategy or mixed strategy? - an initial comparison of their asymptotic convergence rate and asymptotic hitting time. In Jin-Kao Hao and Martin Middendorf, editors, *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2012)*, volume 7245 of *LNCS*, pages 218–229, Malaga, Spain, 2012. Springer.

[99] Jun He, Wei Hou, Hongbin Dong, and Feidun He. Mixed strategy may outperform pure strategy: An initial study. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2013)*, pages 562–569, Cancun, Mexico, 2013. IEEE Press.

[100] Fernanda Hembecker, Heitor S. Lopes, and Walter Godoy, Jr. Particle swarm optimization for the multidimensional knapsack problem. In Bartlomiej Beliczynski, Andrzej Dzielinski, Marcin Iwanowski, and Bernardete Ribeiro, editors, *Proceedings of the International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2007), Part I*, volume 4431 of *LNCS*, pages 358–365, Warsaw, Poland, 2007. Springer.

[101] Robert Hinterding. Mapping, order-independent genes and the knapsack problem. In *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC 1994)*, pages 13–17, Orlando, Florida, USA, 1994. IEEE Press.

[102] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[103] Libin Hong, John Robert Woodward, Jingpeng Li, and Ender Özcan. Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming. In Krzysztof Krawiec, Alberto Moraglio, Ting Hu, A. Sima Etaner-Uyar, and Bin Hu, editors, *Genetic Programming - 16th European Conference (EuroGP 2013)*, volume 7831 of *LNCS*, pages 85–96, Vienna, Austria, 2013. Springer.

[104] Holger H. Hoos and Thomas Stützle. Satlib: An online resource for research on sat. In I.P.Gent, H.v.Maaren, and T.Walsh, editors, *SAT 2000*. IOS Press, 2000.

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

[106] Bernardo A Huberman, Rajan M. Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997.

[107] Matthew Hyde. *A Genetic Programming Hyper-Heuristic Approach to Automated Packing*. PhD thesis, University of Nottingham, UK, 2010.

[108] Matthew Hyde. One dimensional packing benchmark data sets. Online, 2011. URL `http://www.cs.nott.ac.uk/~mvh/packingresources.shtml`.

[109] Matthew Hyde, Ender Özcan, and Edmund. K. Burke. Multilevel search for evolving the acceptance criteria of a hyper-heuristic. In *Proceedings of the Multidisciplinary International conference on Scheduling: Theory and Applications (MISTA 2009)*, pages 798–801, 2009.

[110] IBM. Ibm cplex optimizer. Online, 2013. URL `www.ibm.com/software/commerce/optimization/cplex-optimizer/`.

[111] Atsuko Ikegami and Akira Niwa. Subproblem-centric model and approach to the nurse scheduling problem. *Mathematical Programming*, 97(3):517–541, 2003.

[112] Warren G. Jackson, Ender Özcan, and John H. Drake. Late acceptance-based selection hyper-heuristics for cross-domain heuristic search. In Yaochu Jin and Spencer Angus Thomas, editors, *Proceedings of the 13th Annual Workshop on Computational Intelligence (UKCI 2013)*, pages 228–235, Surrey, UK, 2013. IEEE Press.

[113] Thomas Jansen and Ingo Wegener. Real royal road functions - where crossover provably is essential. *Discrete Applied Mathematics*, 149(1-3):111–125, 2005.

[114] Terry Jones. Crossover, macromutation, and population-based search. In Larry J. Eshelman, editor, *Proceedings of the International Conference on Genetic Algorithms (ICGA 1995)*, pages 73–80, Pittsburgh, PA, USA, 1995. Morgan Kaufmann.

[115] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[116] Murat Kalender, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. A greedy gradient-simulated annealing hyper-heuristic for a curriculum-based course timetabling problem. In *Proceedings of the 12th Annual Workshop on Computational Intelligence (UKCI 2012)*, pages 1–8, Edinburgh, UK, 2012.

[117] Murat Kalender, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. A greedy gradient-simulated annealing selection hyper-heuristic. *Soft Computing*, 17(12): 2279–2292, 2013.

[118] Robert E. Keller and Riccardo Poli. Linear genetic programming of metaheuristics. In Hod Lipson, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, pages 1753–1753, London, UK, 2007. ACM.

[119] Robert E. Keller and Riccardo Poli. Linear genetic programming of parsimonious metaheuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 4508–4515, Singapore, 2007. IEEE Press.

[120] Graham Kendall and Mazlan Mohamad. Channel assignment optimisation using a hyper-heuristic. In *Proceedings of the IEEE Conference on Cybernetic and Intelligent Systems (CIS 2004)*, pages 791–796, Singapore, 2004.

[121] James Kennedy. *Encyclopaedia of Machine Learning*, chapter Particle swarm optimization, pages 760–766. Springer, 2010.

[122] Ahmed Kheiri and Ender Özcan. A hyper-heuristic with a round robin neighbourhood selection. In Martin Middendorf and Christian Blum, editors, *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2013)*, volume 7832 of *LNCS*, pages 1–12, Vienna, Austria, 2013. Springer.

[123] Sami Khuri, Thomas Bäck, and Jörg Heitkötter. The zero/one multiple knapsack problem and genetic algorithms. In *Proceedings of the ACM Symposium on Applied Computing (SAC '94)*, pages 188–193, Phoenix, AZ, USA, 1994. ACM.

[124] Berna Kiraz, A. Şima Uyar, and Ender Özcan. An investigation of selection hyper-heuristics in dynamic environments. In Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Anikó Ekárt, Anna Esparcia-Alcázar, Juan Julián, Merelo Guervós, Ferrante Neri, Mike Preuss, Hendrik Richter, Julian Togelius, and Georgios N. Yannakakis, editors, *Proceedings of the International Conference on the Applications of Evolutionary Computation (EvoApplications 2011)*, volume 6624 of *LNCS*, pages 314–323, Torino, Italy, 2011. Springer.

[125] Berna Kiraz, A. Şima Etaner-Uyar, and Ender Özcan. An ant-based selection hyper-heuristic for dynamic environments. In Anna Isabel Esparcia-Alcázar, editor, *Proceedings of the International Conference on the Applications of Evolutionary Computation (EvoApplications 2013)*, volume 7835 of *LNCS*, pages 626–625, Vienna, Austria, 2013. Springer.

[126] Berna Kiraz, A. Sima Uyar, and Ender Özcan. Selection hyper-heuristics in dynamic environments. *Journal of the Operational Research Society*, 64(12):1753–1769, 2013.

[127] Scott Kirkpatrick, C. Daniel Gelatt Jr, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[128] Muhammet Köle, Sima Etaner-Uyar, Berna Kiraz, and Ender Özcan. Heuristics for car setup optimisation in torcs. In *Proceedings of the 12th Annual Workshop on Computational Intelligence (UKCI 2012)*, pages 1–8, Edinburgh, UK, 2012.

[129] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. The MIT Press, Cambridge, MA, 1992.

[130] Eduardo Krempser, Alvaro Fialho, and Helio Barbosa. Adaptive operator selection at the hyper-level. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Proceedings of Parallel Problem Solving from Nature (PPSN 2012), Part II*, volume 7492 of *LNCS*, pages 378–387, Taormina, Italy, 2012. Springer.

[131] Jiri Kubalik. Hyper-heuristic based on iterated local search driven by evolutionary algorithm. In Jin-Kao Hao and Martin Middendorf, editors, *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2012)*, volume 7245 of *LNCS*, pages 148–159, Malaga, Spain, 2012. Springer.

[132] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[133] Andreas Lehrbaum and Nysret Musliu. A new hyperheuristic algorithm for cross domain search problems. In Youssef Hamadi and Marc Schoenauer, editors, *Proceedings of Learning and Intelligent Optimization (LION 2012)*, volume 7219 of *LNCS*, pages 437–442, Paris, France, 2012. Springer.

[134] Per Kristian Lehre and Ender Özcan. A runtime analysis of simple hyper-heuristics: To mix or not to mix operators. In *Proceedings of the 12th workshop on Foundations of Genetic Algorithms (FOGA XII 2013)*, pages 97–104, Adelaide, Australia, 2013. ACM.

[135] Eunice López-Camacho, Hugo Terashima-Marín, and Peter Ross. A hyper-heuristic for solving one and two-dimensional bin packing problems. In Pier Luca Lanzi Natalio Krasnogor, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*, pages 257–258, Dublin, Ireland, 2011. ACM.

[136] Manuel López-Ibáñez and Thomas Stützle. The automatic design of multi-objective ant colony optimisation algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012.

[137] Manuel López-Ibáñez and Thomas Stützle. An experimental analysis of design choices of multi-objective ant colony optimization algorithms. *Swarm Intelligence*, 6(3):207–232, 2012.

[138] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

[139] James Lorie and Leonard J. Savage. Three problems in capital rationing. *Journal of Business*, 28(4):229–239, 1955.

[140] Helena Ramalhino Lourenço, Olivier Martin, and Thomas Stützle. *Handbook of Metaheuristics*, chapter Iterated Local Search, pages 321–353. Kluwer Academic Publishers, 2003.

[141] Helena Ramalhino Lourenço, Olivier Martin, and Thomas Stützle. *Handbook of Metaheuristics 2nd ed.*, chapter Iterated Local Search: Framework and Applications, pages 363–397. Springer, 2010.

[142] M. J. Magazine and Osman Oguz. A heuristic algorithm for the multidimensional

zero-one knapsack problem. *European Journal of Operational Research*, 16(3):319–326, 1984.

[143] Vittorio Maniezzo, Thomas Stützle, and Stefan Voss, editors. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer, Norwell, MA, USA, 2010.

[144] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1): 50–60, 1947.

[145] Renata Mansini and M. Grazia Speranza. Coral: An exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing*, 24(3):399–415, 2012.

[146] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John WIley & Sons, 1990.

[147] Franco Mascia and Thomas Stützle. A non-adaptive stochastic local search algorithm for the chesc 2011 competition. In Youssef Hamadi and Marc Schoenauer, editors, *Proceedings of Learning and Intelligent Optimization (LION 2012)*, volume 7219 of *LNCS*, pages 101–114, Paris, France, 2012. Springer.

[148] Jorge Maturana, Álvaro Fialho, Frédéric Saubion, Marc Schoenauer, and Michèle Sebag. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 365–372, Trondheim, Norway, 2009. IEEE Press.

[149] Jorge Maturana, Frédéric Lardeux, and Frédéric Saubion. Autonomous operator management for evolutionary algorithms. *Journal of Heuristics*, 16(6):881–909, 2010.

[150] Kent McClymont and Edward C. Keedwell. Markov chain hyper-heuristic (MCHH): an online selective hyper-heuristic for multi-objective continuous problems. In Pier Luca Lanzi Natalio Krasnogor, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*, pages 2003–2010, Dublin, Ireland, 2011. ACM.

[151] M. Misir, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. An investigation on the generality level of selection hyper-heuristics under different empirical conditions. *Applied Soft Computing*, 13(7):3335–3353, 2013.

[152] Mustafa Misir. *Intelligent Hyper-heuristics: A Tool for Solving Generic Optimisation Problems*. PhD thesis, Department of Computer Science, KU Leuven, 2012.

[153] Mustafa Misir, Wim Vancroonenburg, Katja Verbeeck, and Greet Vanden Berghe. A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete. In L. D. Gaspero, A. Schaerf, and T. Stützle, editors, *Proceedings of the Metaheuristics International Conference (MIC 2011)*, pages 289–298, Udine, Italy, 2011.

[154] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. A new hyper-heuristic implementation in hyflex: a study on generality. In *Proceedings of the Multidisciplinary International conference on Scheduling: Theory and Applications (MISTA 2011)*, pages 374–393, 2011.

[155] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. An intelligent hyper-heuristic framework for chesc 2011. In Youssef Hamadi and Marc Schoenauer, editors, *Proceedings of Learning and Intelligent Optimization (LION 2012)*, volume 7219 of *LNCS*, pages 461–466, Paris, France, 2012. Springer.

[156] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. The effect of the set of low-level heuristics on the performance of selection hyper-heuristics. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Proceedings of Parallel Problem Solving from Nature (PPSN 2012), Part II*, volume 7492 of *LNCS*, pages 408–417, Taormina, Italy, 2012. Springer.

[157] Melanie Mitchell, Stephanie Forrest, and John H. Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In Francisco J. Varela and Paul Bourgine, editors, *Proceedings of the First European Conference on Artificial Life*, pages 245–254, Paris, France, 1992. MIT Press.

[158] Elizabeth Montero, Maria-Cristina Riff, and Bertrand Neveu. A beginner's guide to tuning methods. *Applied Soft Computing*, 17(1):39–51, 2014.

[159] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report, Caltech concurrent computation program, C3P Report 826, 1989.

[160] Pablo Moscato, Carlos Cotta, and Alexandre Mendes. Memetic algorithms. In *New optimization techniques in engineering*. Springer, 2004.

[161] Heinz Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In Gregory J. E. Rawlins, editor, *Proceedings of Foundations of Genetic Algorithms (FOGA 1990)*, pages 316–337, Bloomington Campus, Indiana, USA, 1991. Morgan Kaufmann.

[162] Alexander Nareyek. *Metaheuristics: computer decision-making*, chapter Choosing Search Heuristics by Non-Stationary Reinforcement Learning, pages 523–544. Kluwer Academic Publishers, 2001.

[163] Mladenović Nenad and Hansen Pierre. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[164] Su Nguyen, Mengjie Zhang, and Mark Johnston. A genetic programming based hyper-heuristic approach for combinatorial optimisation. In Natalio Krasnogor and Pier Luca Lanzi, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*, pages 1299–1306, Dublin, Ireland, 2011. ACM.

[165] Gabriela Ochoa and Matthew Hyde. The cross-domain heuristic search challenge (CHeSC 2011). Online, 2011. URL `http://www.asap.cs.nott.ac.uk/chesc2011/`.

[166] Gabriela Ochoa, Matthew Hyde, Tim Curtois, José Antonio Vázquez-Rodríguez, James D. Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J. Parkes, Sanja Petrovic, and Rong Qu. Hyflex: A benchmark framework for cross-domain heuristic search. In Jin-Kao Hao and Martin Middendorf, editors, *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2012)*, volume 7245 of *LNCS*, pages 136–147, Malaga, Spain, 2012. Springer.

[167] Gabriela Ochoa, James D. Walker, Matthew Hyde, and Tim Curtois. Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Proceedings of Parallel Problem Solving from Nature (PPSN 2012), Part II*, volume 7492 of *LNCS*, pages 418–427, Taormina, Italy, 2012. Springer.

[168] Mattais Ohlsson, Carsten Peterson, and Bo Söderberg. Neural networks for optimization problems with inequality constraints: the knapsack problem. *Neural Computing*, 5(2):331–339, 1993. ISSN 0899-7667.

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

[170] Ender Özcan and Can Basaran. A case study of memetic algorithms for constraint optimization. *Soft Computing*, 13(8-9):871–882, 2009. ISSN 1432-7643.

[171] Ender Özcan and Ahmed Kheiri. *Computer and Information Sciences II: 26th International Symposium on Computer and Information Sciences*, chapter A Hyper-heuristic based on Random Gradient, Greedy and Dominance, pages 404–409. Springer, 2011.

[172] Ender Özcan and Andrew J. Parkes. Policy matrix evolution for generation of heuristics. In Natalio Krasnogor and Pier Luca Lanzi, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*, pages 2011–2018, Dublin, Ireland, 2011. ACM.

[173] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. Hill climbers and mutational heuristics in hyperheuristics. In Thomas Philip Runarsson, Hans-Georg Beyer, Edmund K. Burke, Juan J. Merelo Guervós, L. Darrell Whitley, and Xin Yao, editors, *Proceedings of the International Conference on Parallel Problem Solving From Nature (PPSN 2006)*, volume 4193 of *LNCS*, pages 202–211, Reykjavik, Iceland, 2006. Springer.

[174] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.

[175] Ender Özcan, Yuri Bykov, Murat Birben, and Edmund K. Burke. Examination timetabling using late acceptance hyper-heuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 997–1004, Trondheim, Norway, 2009. IEEE Press.

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

[177] Andrew J. Parkes, Ender Özcan, and Matthew Hyde. Matrix analysis of genetic programming mutation. In Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado, and Carlos Cotta, editors, *Genetic Programming - 15th European Conference (EuroGP 2012)*, volume 7244 of *LNCS*, pages 158–169, Malaga, Spain, 2012. Springer.

[178] Clifford C. Petersen. Computational experience with variants of the balas algorithm applied to the selection of r&d projects. *Management Science*, 13(9):736–750, 1967.

[179] D. T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi. The bees algorithmŰa novel tool for complex optimisation problems. In Duc T. Pham, Eldaw E. Eldukhri, and Anthony J. Soroka, editors, *Proceedings of the International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2006)*, pages 454–461. Elsevier, 2006.

[180] Hasan Pirkul. A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics*, 34(2):161–172, 1987.

[181] Jakob Puchinger, Günther R. Raidl, and Ulrich Pferschy. The core concept for the multidimensional knapsack problem. In J. Gottlieb and G. R. Raidl, editors, *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2006)*, volume 3906 of *LNCS*, pages 195–208, Budapest, Hungary, 2006. Springer.

[182] Jakob Puchinger, Günther R. Raidl, and Ulrich Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, 2010. ISSN 1526-5528.

[183] Fubin Qian and Rui Ding. Simulated annealing for the 0/1 multidimensional knapsack problem. *Numerical Mathematics*, 16(4):320–327, 2007.

[184] Günther R. Raidl. An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In *Proceedings of the IEEE Conference on Evolutionary Computation (CEC 1998)*, pages 207–211, Anchorage, AK, USA, 1998. IEEE Press.

[185] Günther R. Raidl and Jens R. Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, 13(4):441–475, 2005.

[186] Günther R. Raidl and Jakob Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In Christian Blum, Maria J. Blesa Aguilera, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics*, volume 114 of *Studies in Computational Intelligence*, pages 31–62. Springer, 2008.

[187] Gerhard Reinelt. Tsplib, a library of sample instances for the tsp. Online, 2008. URL `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

[188] Zhilei Ren, He Jiang, Jifeng Xuan, and Zhongxuan Luo. Hyper-heuristics with low level parameter adaptation. *Evolutionary Computation*, 20(2):189–227, 2012.

[189] Andrea Rendl, Matthias Prandtstetter, Gerhard Hiermann, Jakob Puchinger, and Günther R. Raidl. Hybrid heuristics for multimodal homecare scheduling. In

Nicolas Beldiceanu, Narendra Jussien, and Eric Pinson, editors, *Proceedings of Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems (CPAIOR 2012)*, volume 7298 of *LNCS*, pages 339–355, Nantes, France, 2012. Springer.

[190] Peter Ross. Hyper-heuristics. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Technologies*, chapter 17, pages 529–556. Springer, 2005.

[191] Conor Ryan, J.J. Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the European Workshop on Genetic Programming (EuroGP 1998)*, volume 1391 of *LNCS*, pages 83–95, Paris, France, 1998. Springer.

[192] Nasser R. Sabar, Masri Ayob, Rong Qu, and Graham Kendall. A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence*, 37(1):1–11, 2012.

[193] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3-4):591–611, 1965.

[194] Kevin Sim, Emma Hart, and Ben Paechter. Learning to solve bin packing problems with an immune inspired hyper-heuristic. In *Proceedings of the 12th European Conference on Artificial Life (ECAL 2013)*. MIT Press, 2013.

[195] Kevin Sim, Emma Hart, and Ben Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation*, 2014. To appear.

[196] SINTEF. Vrptw benchmark problems, on the sintef transport optimisation portal. Online, 2012. URL `http://www.sintef.no/Projectweb/TOP/VRPTW/`.

[197] Zilla Sinuany-Stern and Ittai Weiner. The one dimensional cutting stock problem using two objectives. *The Journal of the Operational Research Society*, 45(2):231–236, 1994.

[198] Pieter Smet, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Modelling and evaluation issues in nurse rostering. *Annals of Operations Research*, 2012. To appear.

[199] Kenneth Sörensen and Fred Glover. *Encyclopedia of Operations Research and Management Science*, chapter Metaheuristics, pages 960–970. Springer, 2013.

[200] Jerry Swan, Ender Özcan, and Graham Kendall. Hyperion - a recursive hyper-heuristic framework. In Carlos A. Coello Coello, editor, *Proceedings of Learning and Intelligent Optimization (LION 2011)*, volume 6683 of *LNCS*, pages 616–630, Rome, Italy, 2011. Springer.

[201] Gilbert Syswerda. Uniform crossover in genetic algorithms. In J. David Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, Fairfax, Virginia, USA, 1989. Morgan Kaufmann. ISBN 1-55860-006-3.

[202] Eric Taillard. Flow shop benchmark data sets. Online, 2010. URL `http://mistic.heig-vd.ch/taillard/`.

[203] Jorge Tavares, Francisco Baptista Pereira, and Ernesto Costa. Multidimensional knapsack problem: a fitness landscape analysis. *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, 38(3):604–616, 2008.

[204] Paolo Toth and Daniele Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1-3):487–512, 2002.

[205] Michel Vasquez and Jin Hao. A hybrid approach for the 0-1 multidimensional knapsack problem. In Bernhard Nebel, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 328–333, Seattle, Washington, USA, 2001. Morgan Kaufmann.

[206] Michel Vasquez and Yannick Vimont. Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1):70–81, 2005.

[207] Yannick Vimont, Sylvain Boussier, and Michel Vasquez. Reduced costs propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem. *Journal of Combinatorial Optimisation*, 15(2):165–178, 2008.

[208] A. Volgenant and J. A. Zoon. An improved heuristic for multidimensional 0-1 knapsack problems. *Journal of the Operational Research Society*, 41(1):963–970, 1990.

[209] James Walker, Gabriela Ochoa, Michel Gendreau, and Edmund K. Burke. A vehicle routing domain for the hyflex hyper-heuristics framework. In Youssef Hamadi and Marc Schoenauer, editors, *Proceedings of Learning and Intelligent Optimization (LION 2012)*, volume 7219 of *LNCS*, pages 265–276, Paris, France, 2012. Springer.

[210] Richard A. Watson and Thomas Jansen. A building-block royal road where crossover is provably essential. In Hod Lipson, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, pages 1452–1459, London, UK, 2007. ACM.

[211] Tony Wauters, Wim Vancroonenburg, and Greet Vanden Berghe. A guide-and-observe hyper-heuristic approach to the eternity ii puzzle. *Journal of Mathematical Modelling and Algorithms*, 11(3):217–233, 2012.

[212] H. Martin Weingartner and David N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15(1):83–103, 1967.

[213] Christophe Wilbaut and Saïd Hanafi. New convergent heuristics for 0-1 mixed integer programming. *European Journal of Operational Research*, 195(1):62–74, 2009.

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