Novel Hyper-heuristics Applied to the Domain of Bin Packing

Kevin Sim

Institute for Informatics and Digital Innovation

Edinburgh Napier University

A thesis submitted in partial fulfilment of the requirements of Edinburgh Napier University, for the award of Doctor of Philosophy

October 2014

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification at this or any other University or learning institution.

External examiner Prof. David Wolfe Corne

Internal examiner Dr. Neil Urquhart

Director of studies Prof. Emma Hart

Second supervisor Prof. Ben Paechter

[Kevin Sim]

Copyright

Abstract

Principal to the ideology behind hyper-heuristic research is the desire to increase the level of generality of heuristic procedures so that they can be easily applied to a wide variety of problems to produce solutions of adequate quality within practical timescales. This thesis examines hyper-heuristics within a single problem domain, that of Bin Packing where the benefits to be gained from selecting or generating heuristics for large problem sets with widely differing characteristics is considered. Novel implementations of both selective and generative hyper-heuristics are proposed. The former approach attempts to map the characteristics of a problem to the heuristic that best solves it while the latter uses Genetic Programming techniques to automate the heuristic design process. Results obtained using the selective approach show that solution quality was improved significantly when contrasted to the performance of the best single heuristic when applied to large sets of diverse problem instances. Although enforcing the benefits to be gained by selecting from a range of heuristics the study also highlighted the lack of diversity in human designed algorithms. Using Genetic Programming techniques to automate the heuristic design process allowed both single heuristics and collectives of heuristics to be generated that were shown to perform significantly better than their human designed counterparts. The thesis concludes by combining both selective and generative hyper-heuristic approaches into a novel immune inspired system where heuristics that cover distinct areas of the problem space are generated. The system is shown to have a number of advantages over similar cooperative approaches in terms of its plasticity, efficiency and long term memory. Extensive testing of all of the hyper-heuristics developed on large sets of both benchmark and newly generated problem instances enforces the utility of hyper-heuristics in their goal of producing fast understandable procedures that give good quality solutions for a range of problems with widely varying characteristics.

Acknowledgements

My thanks to Professor Emma Hart for her guidance and inspiration before and throughout the period of study culminating in this thesis. Also to Professor Ben Paechter for providing additional input and support throughout the course of the project. To my wife Sharon and children Emma and Liam my thanks for their encouragement, support, patience and understanding over the past years.

To Sharon, Emma, Liam and Isaac.

Contents

List of Figures

List of Tables

List of Definitions

List of Equations

Chapter 1

Introduction

Ever since the addition of computers to the toolbox of scientists, practitioners have revelled in their ability to process vast amounts of structured information in a fraction of the time that a human brain can manage. Early visionaries envisaged machines with immense processing power that would potentially be able to solve the most complex and intractable of problems. Over half a century later this vision has not materialised and whilst computational devices with immense power are everyday items for most people their ability to tackle what initially appear as ideal problems to be solved using a *thinking* machine have not become a reality. Although computational power has continued to grow in accordance with Moore's law, doubling in size every 2 years, developing complete problem solvers for even the *simplest* of combinatorial problems is still well outside their scope for all but the smallest of problem instances. The task of finding solutions to these types of problem has fallen to the computer scientist who, inspired by the wonders of the natural world have strived to replicate the desirable traits inherent in many naturally occurring processes and harness them for their computational needs.

The ever increasing numbers of related scientific papers emerging from academia highlights the continued interest and potential benefits associated with finding fast, robust problem solvers for combinatorial problems. There is however little correlation between the increasing amount of research conducted and the uptake of these potentially financially rewarding techniques by industry. This is often cited as being due to

the complexity and unpredictability associated with stochastic search techniques that typically require extensive tuning and evaluation by experts in order to be applied to specific industrial applications. There is a gap between academic research and the real world constraints that are faced by industry that researchers are in part failing to address. Many modern search techniques are too complex to be adapted for real world problems where the number of constraints faced is the predominant factor. Academics often evaluate their highly specialised methods on unrealistic, over simplistic and contrived benchmark problem sets which although useful as a means of contrast is of little practical use to industry and non-experts with little tangible benefit for society.

Hyper-heuristics are a relatively new collection of diverse approaches that have predominately been researched during the last decade and a half. Amongst the goals set out by early hyper-heuristic practitioners was the ability to provide simple and understandable problem solvers that were easily adaptable and applicable to a range of different problems with little or no modification. Current hyper-heuristic methods are, in general, becoming increasingly more complex and the community could be criticised as straying from the original vision. This trend can be seen with other biologically inspired fields that have emerged over previous decades, such as in the metaheuristic community which started with the promise of providing general problem solvers but has, maybe inevitably, evolved into one that is predominately filled with academics jostling for position in an continually evolving race to provide the best possible solutions to sets of intractable, yet often fabricated and unrealistic benchmark problems.

The work presented here can be said to suffer from many of these criticisms but does not set out to provide perfect solutions. The research presented attempts to highlight the *potential* of hyper-heuristics by conducting an investigation into their use within a single, relatively simple problem domain; the bin packing domain, specifically the the off-line variant of the fixed capacity one dimensional bin packing problem.

Furthermore the underlying search space, defined by a set of heuristics, is constituted of only deterministic heuristics (or their component parts), allowing for greater confidence when analysing the performance of the overseeing hyper-heuristic than could be inferred by searching over sets of stochastic heuristics. This thesis concen-

trates on investigating the benefits to be gained by using hyper-heuristics to exploit the combined utility of sets of deterministic heuristics. Both selective and generative approaches are presented and evaluated on a large corpus of problem instances, both taken from the literature and newly generated.

Stochastic search techniques are often criticised as providing no guarantee of solution quality are are often highly specialised, performing well on only those problem instances that they were designed or evolved to solve. They are often also very costly in terms of design and execution times. In contrast the procedures emerging from this study are simple deterministic procedures that have guaranteed performance and short execution times and are easily understandable by non experts.

The literature suggests that hyper-heuristics can be applied successfully to many classes of problem, possibly even being able to transcend domain boundaries. However, even within a single domain, problem instances can be categorised as belonging to a particular *class* of problem that is best suited for solving using a particular method. The research presented shows that different heuristics are best suited to problem instances generated with particular characteristics, that can be considered as problem classes from the perspective of a heuristic. Although not easily identified these correlations are exploited by the hyper-heuristics introduced. The study conducted allows for the overseeing hyper-heuristics to be evaluated without the uncertainties associated with stochastic heuristics or the complications and inevitable generalisations that occur when hyper-heuristics are applied to different domains.

The following section outlines the main research questions that have driven the research conducted and presented in this thesis.

1.1 Research Questions

- Question 1. To what extent can a deterministic constructive heuristic's ability for solving a problem be mapped to a problem's characteristics and therefore be exploited by a selective hyper-heuristic?

- Question 2. To what extent can novel heuristics be evolved that match or outperform human-designed deterministic constructive heuristics?

- Question 3. Can a hyper-heuristic be used to manage a collective of automatically generated heuristics that collaborate to efficiently cover large problem spaces composed of problems of differing characteristics?

The initial few chapters of this thesis concentrate on investigating which of a range of characteristics best describe a BPP instance and examining whether a correlation can be found between those characteristics and the solution quality obtained by a particular heuristic. The extent to which collectives of heuristics, which can be intelligently selected from, can improve on their individual utility over large problem sets is investigated. Subsequent chapters show the extent to which automatically generated heuristics, both individually and as a collective, can improve upon the quality of the solutions attained by human designed heuristics.

1.2 Thesis Contribution

This thesis explores the field of hyper-heuristics in a single problem domain, that of bin packing. The main contributions of this thesis, listed in increasing order of significance, are listed below.

- The introduction of a new deterministic heuristic for the BPP (Section 3.4.7) and two new sets of benchmark problem instances (Section 3.3.7). The new heuristic is show to perform better than the other man made heuristics investigated when applied to problem instances where the average item weight was small in relation to the bin capacity.
- The introduction of a novel selective hyper-heuristic (Chapter 4) which selects the best heuristic for a problem instance based on characteristics of the problem space that are autonomously derived during an off-line training phase.
- A novel application of genetic programming as a generative hyper-heuristic for generating constructive deterministic heuristics for the BPP (Chapter 5). Unlike other approaches in the literature the nodes combined using Single Node Genetic Programming incorporate mechanisms to explicitly pack items into a solution.

- Two novel approaches for generating sets of cooperative heuristics including the application of an island model and an artificial immune system as a hyper-heuristic (Chapter7). Both methods use the novel concept of generating cooperative sets of heuristics concurrently that better cover the problem space (Chapters 6 and 7). A side effect of the AIS inspired approach is the ability to summarise large problem spaces using small sets of problem instances that encapsulate the entire problem space without loss of information. (Chapter 7)

1.3 Publications Resulting from the Period of Study

The following conference papers, competition entries, book chapters and journal articles, listed in chronological order, were published during the course of the period of study resulting in this thesis.

- Kevin Sim. Ksats-hh: A simulated annealing hyper-heuristic with reinforcement learning and tabu-search. Entry in the Cross-domain Heuristic Search Challenge, June 2011.

- Kevin Sim. Asynchronous idiotypic network simulator. In Emma Hart, Jon Timmis, Paul Mitchell, Takadash Nakamo, Foad Dabiri, Ozgur Akan, Paolo Bellavista, Jiannong Cao, Falko Dressler, Domenico Ferrari, Mario Gerla, Hisashi Kobayashi, Sergio Palazzo, Sartaj Sahni, Xuemin (Sherman) Shen, Mircea Stan, Jia Xiaohua, Albert Zomaya, and Geoffrey Coulson, editors, *Bio-Inspired Models of Networks, Information, and Computing Systems*, volume 103 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 248–251. Springer Berlin Heidelberg, 2012.

- Emma Hart and Kevin Sim. *Computational Intelligence*, chapter Artificial Immune Algorithms in Learning and Optimisation. UNESCO Encyclopedia of Life Support Systems (EOLSS), 2012.

- Kevin Sim, Emma Hart, and Ben Paechter. A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by

attribute evolution. In CarlosA.Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 348–357. Springer Berlin Heidelberg, 2012.

- Kevin Sim and Emma Hart. Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In Christian Blum, editor, *GECCO '13: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, New York, NY, USA, 2013. ACM.

- Kevin Sim, Emma Hart, and Ben Paechter. Learning to solve bin packing problems with an immune inspired hyper-heuristic. In Giuseppe Nicosia Stefano Nolfi Pietro Lió, Orazio Miglino and Mario Pavone, editors, *Advances in Artificial Life, ECAL 2013: Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems*, pages 856–863. MIT Press, 2013.

  K. Sim, E. Hart, and B. Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation Journal*, In Press, January 2014.

1.4 Thesis Layout

The remainder of the thesis is structured as follows:

- Chapter 2 Background : Introduces the one dimensional bin packing problem and the field of hyper-heuristics, concentrating on approaches that have been applied to the bin packing problem.

- Chapter 3 Benchmark Heuristics and Problem Instances : Introduces the benchmark problem instances and heuristics for the BPP that are used throughout this thesis.

- Chapter 4 Selective Hyper-heuristics : A novel selective hyper-heuristic is presented that uses classification techniques to predict the most suitable heuristic

from a pool of benchmark heuristics for a problem instance based on characteristics of the problem instance.

- Chapter 5 Generative Hyper-Heuristics: A generative hyper-heuristic is introduces which uses a form of genetic programming to evolve deterministic heuristics for the BPP that are tested on a large set of benchmark problem instances.
- Chapter 6 Generating Sets of Co-operative Heuristics using a Genetic Programming Island Model : An island model is applied to the generative hyper-heuristic introduced in Chapter 5 to co-evolve sets of deterministic heuristics that collectively cover the heuristic space better than any of the constituent heuristics.
- Chapter 7 An Artificial Immune System Inspired Generative Hyper-heuristic : A novel artificial immune system hyper-heuristic replicates the results obtained using the island model introduced in Chapter 6 using a more efficient approach that is shown to have additional beneficial properties such as memory and increased responsiveness.
- Chapter 8 Conclusions : The final chapter summarises the research presented in the thesis, discusses findings and suggests potential avenues for further research.

Chapter 2

Background

2.1 Introduction

The quantity of research in the field of hyper-heuristics has escalated greatly since the term was first introduced at the start of the millennium [31]. Driven by a goal to design simple and understandable methods capable of generalising over a wide range of problems, practitioners of hyper-heuristics have applied a variety of approaches with many diverse methods falling under the hyper-heuristic banner. These range from applying metaheuristic techniques to search for the best combination of perturbative heuristics to generative techniques that are employed to automate the heuristic design process. While it is the view of many practitioners that a hyper-heuristic should be applicable to different problem classes from multiple domains without modification, the research presented in this thesis explores the value of hyper-heuristics when applied to a single problem domain. It is shown that within a single problem domain there are many classes of problem that vary in difficulty from the perspective of any single heuristic, making hyper-heuristics useful as more general solvers even within a single domain. This thesis explores hyper-heuristics in the realm of the One Dimensional Bin Packing domain and presents a number of probabilistic hyper-heuristics that search over a set of deterministic constructive heuristics, both selected from the literature and automatically generated for this relatively simple, yet intractable, combinatorial optimisation problem. Restricting the study to the sub-class of hyper-heuristics that search over

a set of constructive deterministic heuristics allows for an analysis of the merits of hyper-heuristics to be conducted without being affected by the uncertainty inherent in stochastic perturbative heuristics. It is the authors belief that many of the techniques presented in this thesis could easily be transferred to other problem domains but in order to maximise a hyper-heuristics potential, as with other search techniques, each application requires costly and lengthy tuning in order to optimise its performance. The typical view that dominates the literature is that a hyper-heuristic operates at a level of abstraction independent of the problem domain without access to any domain specific knowledge. This however is a generalisation that especially in the case of selective hyper-heuristics does not always hold. Subsequent chapters show that hyper-heuristic approaches can benefit from deriving information specific to the problem instances being solved and that the relationship between a problem instance's characteristics and the ability of a heuristic to provide a good solution can be quantified and exploited to improve the quality of the subsequent solution.

This chapter first introduces the BPP before reviewing some of the hyper-heuristic literature that has introduced a broad range of techniques that have been applied to a range of different combinatorial problems. Subsequent chapters provide additional specific background information that is relevant to the presented hyper-heuristic approach.

2.2 One Dimensional Bin Packing Problem

The Bin Packing Problem (BPP) is a combinatorial optimisation problem which belongs to the larger class of cutting and packing problems that occur frequently in industrial applications. The objective of the problem, of which there are many variants, is to pack a number of items into as few containers (bins) as possible. The problem is widely researched in part due to the fact that it appears as a component part of more complicated real world problems such as vehicle transportation and scheduling tasks. Due to the wealth of research available it is an ideal domain to use as an example to explore the merits of hyper-heuristics.

One method of classifying different variants of BPP is by their dimensionality. The

problem may be to minimise:

- The number of machines required to process a varied set of jobs or the number of data packets needed to transmit a quantity of digital information (One Dimensional).
- The waste from a quantity of sheet material used to cut a set of shapes (Two Dimensional).
- The volume needed to transport a range of of goods of varying dimensions (Three Dimensional).

A second predominant feature of BPPs that affects the way that search methods can approach the problem is the availability of the items that make up the problem instance.

- In the On-Line BPP, items requiring to be packed are presented as they become available. In the theoretical problem, typically these are packed on arrival into containers, all of which remain available for the duration of the procedure.
- In the Off-Line BPP all of the problem instance's items are known *a priori* allowing for pre-processing of the items if desired. In the academic literature the containers used to pack the items that make up an instance of this class of BPP are usually available for the duration of this procedure.

Subsequent chapters of this thesis introduce a number of different hyper-heuristics that search over a range of deterministic constructive heuristics for the fixed capacity variant of the off-line BPP. The underlying heuristics were either designed based on descriptions taken from the literature or were automatically generated. In all cases the underlying heuristics received items in descending order of size and were designed to iteratively pack a single bin at a time which, once filled, is immediately closed and becomes unavailable to the packing procedure. This could be seen as a special case of the off-line BPP where containers need to be dispatched as quickly as possible, such as for a large distribution network where loading space is constrained and driver idle time is a relevant financial factor. All of the deterministic heuristics used throughout

this study require the prerequisite that all of the items to be packed are known prior to the procedure commencing and that those items are presented in descending order of size. However all of the heuristics investigated can also be used on real world variants of BPPs where items are presented continually in batches, as long as each set of items presented meets the constraint that they are pre-ordered into descending size order.

The off-line variant of the one dimensional bin packing problem with equal bin capacity is used here as an example domain with any observations made applying to problems of a higher dimension, varied bin capacity or for problems where additional constraints, such as bin size to cost ratio, are a factor. The abbreviation BPP is used throughout the remainder of this document in relation to this unidimensional class of problem which is defined below.

Definition 2.1 (One-Dimensional Bin Packing Problem (BPP))**.** *The BPP is a class of NP-hard problem [51] the objective of which is to minimise the number of bins of fixed capacity C required to accommodate a set J of n items with $J = \{\omega_1 \dots \omega_n\}$ and weight $\omega_j : j \in \{1 \dots n\}$ falling in the range $1 \leq \omega_j \leq C$ whilst enforcing the constraint that the sum of weights in any bin does not exceed the bin capacity C [104].*

For any problem instance the theoretical optimal number of bins required to pack all of the items in a problem is given by Equation 2.1. There have been many studies conducted in order to refine this measure for a given problem instance where the absolute lower bound is unattainable, most notably [51]. For example if each of a problem instance's items are greater in size than half of the capacity of a bin then each item requires placing into its own bin and clearly the optimal number of bins is simply equal to the number of items in the problem. The objective of hyper-heuristic methods is not to exhaustively search for an optimal solution but to return an adequate solution within a reasonable time-scale. Therefore, although it is clear that the lower bound given by Equation 2.1 is unattainable for many problem instances it is used throughout this thesis wherever the optimal solution is unknown as it provides an absolute lower

bound by which the quality of a solution can be gauged .

$$opt = \left\lceil \frac{\sum_{j=1}^{n} \omega_j}{C} \right\rceil \tag{2.1}$$

As well as a wealth and variety of different techniques that have been applied to the BPP there are also a number of sets of widely used benchmark problem instances that have been used by academics to contrast their implementations and that are readily available via on-line repositories . The most commonly used benchmark problems for the BPP come from two publications. In [104] Scholl, et al., introduced 1210 problem instances split into 3 data sets whilst in a separate publication [42] Falkenauer introduced another 160 widely used problem instances. These problem instances are described in detail in Chapter 3 where the affect of different problem characteristics on the difficulty of the problems from the perspective of a range of human designed deterministic constructive heuristics is investigated.

The BPP has been addressed using a number of techniques

- Exact [104] methods are typically only feasible for small instances of BPP due to the exponential growth in computational resources required with increased problem size.
- Heuristics [51] are usually simple rules of thumb that give good, but not necessarily optimal, performance in terms of both solution quality and execution time. As is shown in subsequent chapters different heuristics work best on certain classes of problems with different characteristics.
- Metaheuristics such as Genetic Algorithms [40] and Ant Colony Optimisation [74] are amongst the most successful approaches that have been applied to the BPP. The two examples cited are both highly tailored hybrid algorithms that incorporate a problem specific local search based on the reduction procedure of Martello and Toth [80].

Hyper-heuristics have also been successfully applied to the BPP. Some of these approaches are included in the review of the hyper-heuristic literature covered in the

following section.

2.3 Hyper-Heuristics

Amongst the goals of hyper-heuristic research is to create more general procedures capable of being applied to different problems of varying characteristics, a goal shared with early metaheuristic practitioners introduced a decade and a half earlier. Even though metaheuristics are amongst the most successful approaches for addressing combinatorial optimisation (CO) problems they are often overlooked by industry, being perceived as overcomplicated, requiring costly and lengthy expert knowledge in order to be tuned to each new problem or as yet unexplored problem domains. Hyper-heuristics aim to reduce this overhead by offering fast and easy to implement techniques capable of providing "good enough - soon enough - cheap enough" solutions to problems from different domains[14].

Unlike much of the research presented in the field of metaheuristics the objective for hyper-heuristics is not primarily to generate procedures capable of solving to optimality a subset of well documented problems for a given domain. Hyper-heuristics aim to provide robust procedures that are easily adaptable and perform satisfactorily in terms of solution quality and execution time for a wider range of problems, even possibly being able to transcend domain boundaries. Hyper-heuristics differ to other search paradigms in that they search over a space defined by a set of problem specific heuristics, or heuristic components rather than directly over the problem space. Hyper-heuristic approaches may search for a suitable heuristic, combinations of simple heuristics or combinations of heuristic components that are used to generate new heuristics.

The term hyper-heuristics was not used in the literature until 1997 where it was applied in the domain of automated theorem proving to describe an amalgamation of artificial intelligence techniques [35]. The concept however, dates back to the 1960's where, for example, Fisher and Thompson [47] used combinations of simple dispatching rules to improve upon the solutions attained using single rules for a range of scheduling problems. First used in relation to combinatorial optimisation problems

at the beginning of the millennium [31], the term hyper-heuristics has evolved to encompass a number of diverse methods that are related by the fact that they search over the space defined by the underlying heuristics rather than the search space defined by the set of possible solutions.

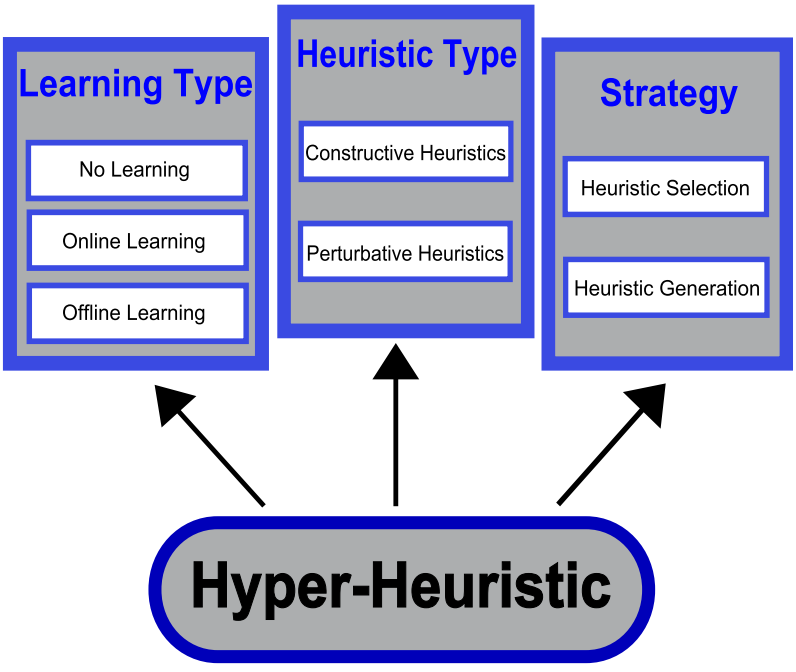
Many hyper-heuristics presented in the literature are evaluated on multiple domains. The hyper-heuristic searches for permutations of simple problem specific heuristics (or their component parts) using a generalised heuristic or metaheuristic process that is unchanged for each domain. Only the underlying heuristics vary across different problem domains with the hyper-heuristic receiving no problem specific information to aid its search. The focus of this thesis differs from this view and explores the realm of hyper-heuristics within the bounds of a single domain with the emphasis on selecting, generating and combining deterministic heuristics specific to the BPP. Information specific to the problem domain and characteristics specific to the problem instances being solved are exploited to improve the search and to provide a memory that can be used to increase the quality and speed of successive searches.

Subsequent chapters show that individual heuristics have limited utility when applied to problem instances of widely varying characteristics. By selecting or generating a heuristic appropriate to the instance, or instances, to be solved the combined utility of a set of problem specific heuristics can be exploited thereby minimising the weaknesses inherent in any deterministic heuristic. Although it is the opinion of the author that the research presented is applicable to other CO problem domains this remains as an objective for future research with the contribution of the thesis being restricted to a single domain. As noted, even within a single problem domain, there is a trade off between the ability of a heuristic to perform well on a specific problem instance, a class of problem instances with similar characteristics and its ability to generalise over a wider set of more varied problem instances.

The problem of finding the best permutation of heuristics in itself a CO problem[120, 121]. However the landscape to be traversed, defined by the set of underlying heuristics, may be easier to navigate than the landscape defined by the possible set of solutions.

A recent classification of the term hyper-heuristics is given in [19] to encompass

emerging techniques in the field. Whilst earlier works concentrated on developing strategies based on the original definition of *"heuristics to select heuristics"* [14] more recent approaches have introduced the concept of *"heuristics to generate heuristics"* [19] typically using genetic programming as a method of creating new heuristics from constituent parts. Burke et al. [19] classify a hyper-heuristic dependent upon which subclass of three defining categories the approach implements, as depicted in figure 2.1, giving a total of 12 possible classifications. However, whilst the categories themselves are clear, attempting to classify the papers listed in a comprehensive bibliography [86] as belonging to one of the 12 categories is not always possible due to the hybrid approach taken by many practitioners. Subsequent sections of this chapter review a number of hyper-heuristics from the literature which are separated, as far as possible, based on which of the primary underlying two strategies they use; Selective or Generative.



Heuristics have been defined as belonging to a number of different categories As an example a hyper-heuristic could be classified as a heuristic selection strategy using constructive heuristics with no learning.

Figure 2.1: Classification of hyper-heuristic approaches

The descriptions given in definitions 2.2 - 2.10 clarify the terms used in the classification described in [19] and depicted in figure 2.1.

Definition 2.2 (Hyper-heuristic)**.** *The term **hyper-heuristic** is deemed to mean any approach in which the primary search is conducted over a space defined by a pool of underlying heuristics or their component parts. A hyper-heuristic searches this space looking for a suitable method for solving one or more problem instances. This is achieved by selecting, generating, combining or altering elements taken from the underlying pool of heuristics or heuristic components. A hyper-heuristic may be used to create novel reusable heuristics that can be applied without modification to new problem instances or it may operate adaptively during the course of solving one or more problem instances changing its behaviour in relation to the* state *of the complete or partial solution.*

Definition 2.3 (Heuristic)**.** *A **heuristic** may be a constructive procedure that is used to build a solution or a perturbative operator that acts upon an already complete solution. The output from a heuristic is a complete solution to the problem being tackled.*

Definition 2.4 (Heuristic Component)**.** *A **component** may be a parameter or method within an algorithm that if altered affects the algorithms performance. In the context of generative hyper-heuristics the components may be arithmetic operators, problem specific variables or procedures which when combined result in a new heuristic. When used as building blocks constructive heuristics can be defined as components and the resultant combination of components (found by applying a hyper-heuristic) can be seen as a new heuristic that can be reused without further modification.* [1]

The clearest defining feature of different hyper-heuristic approaches is whether they select from a set of pre-determined heuristics, or heuristic components, or whether they generate new heuristics from simpler constituent parts.

[1] Many deterministic constructive heuristics described in this thesis are created by combining *components*. Many of these components can be applied iteratively to a problem instance in order to build a complete solution. If used in isolation the components can be defined as a heuristic. If combined they become components of a new heuristic.

Definition 2.5 (Selective Hyper-heuristic)**.** *A **selective hyper-heuristic** is responsible for selecting which heuristic(s) to apply to construct or perturb a solution for one or more problem instances. This could range from applying a single heuristic to generate a complete solution to deciding the order that a range of perturbative heuristics are applied in order to improve upon an incumbent. Typically hyper-heuristics adopt both a selection strategy and an acceptance strategy in order to make this decision, deciding which heuristic to apply and also whether to accept the resultant solution.*

Definition 2.6 (Generative Hyper-heuristic)**.** *Generative hyper-heuristics aim to automate the heuristic design process by combining heuristic components to create new heuristics. The components may include mathematical operators and problem specific variables that are constructed into functions used to determine actions such as the choice of item and the choice of item placement in a solution. Components could also include programmatic elements or partial heuristics that are perturbed or combined in order to create new heuristics.*

The heuristics that are either selected or generated using a hyper-heuristic can be categorized as being either constructive of perturbative.

Definition 2.7 (Constructive Heuristic)**.** *A constructive heuristic builds a solution from an initially empty solution, controlling the order that items are selected and placed into the solution. At each iteration the heuristic is responsible for determining which item(s) from the problem instance are selected for placement into the partial solution as well as the location at which they should appear. Constructive heuristics have a natural termination point when all items have been inserted.*

Definition 2.8 (Perturbative Heuristic)**.** *A perturbative heuristic works to improve upon an already complete solution by perturbing the order of items using, for example, mutation operators or hill climbing algorithms. Perturbative, hyper-heuristics start with an initial complete solution that each underlying heuristic can operate upon to create a new solution within that heuristics own local neighbourhood. Unlike constructive approaches perturbative approaches have no natural termination point and are typically executed for a predefined time limit or number of iterations or until the quality of the solution is deemed sufficient.*

Definition 2.9 (Learning)**.** *A hyper-heuristic that uses **No Learning** uses a predefined strategy with no adaptation during the search process. As example a hyper-heuristic which uses simulated annealing as the guiding strategy retains no memory of the search history and uses the same predefined acceptance strategy throughout the lifetime of the algorithm.*

Definition 2.10 (Hyper-heuristic with Learning)**.** *A **learning** hyper-heuristic is a hyper-heuristic that uses memory either explicitly such as is the case with tabu search or learning classifier systems or implicitly as is the case with evolutionary approaches such as an EA in order to improve the search based on previous history.*

- ***On line** learning uses information acquired during the current search which is not stored between algorithm restarts. Heuristics emerging from this approach could be considered as disposable in the context that they are disregarded after use.*
- ***Off line** learning uses memory obtained during previous searches, typically conducted during a training phase prior to the algorithm being used in its final context.*

Perturbative hyper-heuristics operates similarly to many metaheuristic techniques, iteratively attempting to improve upon an incumbent solution. The hyper-heuristic is responsible for selecting which of a number of mutation or hill climbing heuristics to apply at each iteration and for determining whether the resultant solution should be accepted. This class of hyper-heuristic is amongst the most popular and most successful techniques to be found in the literature [6, 10, 13, 92] but they can suffer from many of the undesirable traits inherent in metaheuristic techniques. Having no natural termination point means that execution times can be unpredictable with no guarantee of solution quality due to the stochastic nature of the underlying heuristics.

In contrast the hyper-heuristics presented in this work search over a set of deterministic constructive heuristics or their components. The resultant procedures are easy to understand, fast to execute and are shown to be effective when applied to a wide range of problem instances in a single domain. The procedures evolved using the hyper-heuristics presented in this thesis are reusable in that they can be stored and reapplied

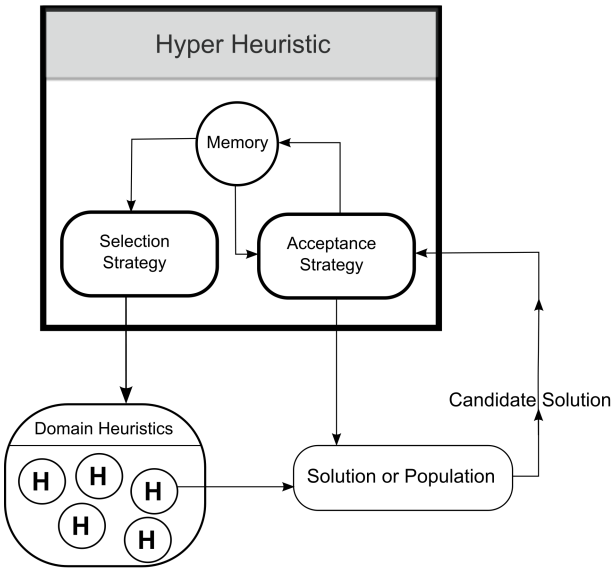
to replicate the solutions attained due to the deterministic nature of the heuristics and heuristic components used. This affords an increased level of assurance in performance of the final procedures over the variable solution quality attained using stochastic heuristics.

A generalised hyper-heuristic model is shown in figure 2.2 which depicts the hyper-heuristic as an overseeing process responsible for selecting which heuristic to apply next in order to either construct or perturb a solution. The hyper-heuristic is responsible for deciding whether to accept the solution in which case it replaces, for a single point search, the incumbent or in the case where a parallel search is being conducted it is added as, or replaces, a solution from the population. Memory may be employed, in one or both of the selection and acceptance strategies, to offer some learning capabilities in order to improve the search based on past history. The hyper-heuristic solution consists of a method to obtain the final problem solution. This may be manifested as a fixed list of heuristics that are applied in order to construct or perturb a solution or as a process or set of rules that is used to determine the selection of heuristics.

The above classification scheme is not exhaustive and there are many search methods which are defined as hyper-heuristics that use a combination of approaches such as in [55] where a heuristic selection strategy is employed to select between constructive, perturbative and noise heuristics using an "evolutionary-based hyperheuristic approach, called EH-DVRP".

A comprehensive account of the origins and evolution of hyper-heuristics is given in [12] which points to works carried out in different combinatorial optimisation problem domains including scheduling [45, 47, 60, 84], packing [78, 101, 102], constraint satisfaction [127], time-tabling[129] and vehicle routing [52]. Other recent overviews are provided in the review papers [93] and [26]. A selection of book chapters have also been published relating to hyper-heuristic research [14, 18, 19, 26, 100].

A number of different guiding techniques have been used in hyper-heuristic implementations ranging from simple local search or hill climbing methods[120, 121], metaheuristic approaches including evolutionary algorithms[45], tabu search [25], simulated annealing [5] and ant colony optimisation [8, 27] as well as machine learning strategies such as learning classifier systems [101] and reinforcement learning [83].



The hyper-heuristic decides which heuristic from a pool of domain specific heuristics to apply to the complete or partial solution before determining whether to accept the modified outcome. Memory may be used, either obtained during the current execution or learned previously, in an attempt to improve the search process. In a population based parallel search the hyper-heuristic is also responsible for determining the selection and replacement of the solution to be modified which is not shown for clarity.

Figure 2.2: Operation cycle of a typical hyper-heuristic

The following section looks at some of the approaches that pre date the term hyper-heuristics being coined followed by a more in depth study of some of the methodologies applied during the last decade, grouped according to the classification scheme proposed in[19], as depicted in figure 2.1, concentrating on those most relevant to the research presented here.

2.3.1 Early Hyper-Heuristic Approaches

In the 1990's, prior to the term hyper-heuristics being coined, with computational power growing, advances in heuristic techniques made by operations research, the appearance of stochastic metaheuristic search techniques and progress in the field of machine learning the number of researchers investigating some of the underlying ideas behind what is now termed hyper-heuristics also increased. Regardless of the dawn of

the expression *hyper-heuristics* and its changing definition the idea of solving computationally hard problems by combining known methodologies is one that dates back to the 1960's when Fisher & Thompson [47] used machine learning techniques to select combinations of simple heuristics to produce solutions to job-shop scheduling problems. The domain of scheduling is the one that attracted researchers interests prior to the technique being formalised. Some of the early approaches applied to a variety of different scheduling problems are briefly reviewed.

In [45] a genetic algorithm was used to search "a space of *abstractions* of solutions" evolving a sequence of constructive heuristics to be applied to the problem instance in order to build a complete solution. The authors termed this method an "Evolving Heuristic Choice (EHC)" citing the early work of others in developing "GA/heuristic hybrids". EHC was found to produce promising results on Taillard's benchmark problems [122] even improving upon best results found previously using Tabu Search.

Heuristics were combined in [60] to solve 12 dynamic job-shop scheduling problem instances taken from [82]. The results presented showed that the technique outperformed 3 more conventional search techniques taken from the literature for a large percentage of problem instances and proved highly competitive on the others.

A number of other *heuristic selection methods* that have been applied to scheduling problems which pre date the formalisation of the term hyper-heuristic include [45, 133]. The remainder of this chapter takes a look at some of the hyper-heuristic approaches published since the millennium attempting to classify them according to the main criteria specified byBurke et al. [19] as illustrated in figure 2.1. A number of selective hyper-heuristics are described in Section 2.3.2 before a selection of generative hyper-heuristics are reviewed in Section 2.3.3.

2.3.2 Heuristic Selection Techniques

A number of selective hyper-heuristics techniques are reviewed with emphasis given to those that have been applied to various packing and cutting problems.

Ross et al. [101] use an off-line learning stage to train a learning classifier systems, namely XCS [134], to predict the best constructive heuristic to use based on the current problem state. This work was extended in [78]. Solutions to the one dimensional

bin packing problem instances that it was tested on are built up iteratively as the classifier monitors the changing problem state deciding which heuristic to apply at any given stage. Off-line training was conducted on a training set comprising of 75% of the problem instances taken from a set of 890 benchmark problem instances sourced from the literature. The system evolved rules that determined which combinations of heuristics to use to construct solutions. Eight simple constructive heuristics were utilised, four of which were taken directly from the literature with each modified by the addition of a filler process to make up the remainder of the set. Results showed that superior quality solutions were obtained using combinations of heuristics than were produced by applying any of the heuristics in isolation. The approach investigated the concept that a relationship exists between the structure of a problem instance and the quality of the solution produced by a particular heuristic. If it is possible to map this relationship then the time taken to construct solutions of acceptable quality could be greatly reduced from, for example, the time taken to solve the problem using a metaheuristic. The relationship between problem characteristics and the utility of different heuristics is covered in detail in the following chapter. The heuristics used were LFD, NFD, DJD and DJT[1] as well as four variations of these which incorporated a filler process that continued to pack items into a bin if any were available. These simple deterministic constructive heuristics are described later in Section 3.4. The system was able to find optimal results for 78.1% of the problem instances used during training and 77.7% of the remaining unseen problem instances. These results were not greatly improved upon the results obtained using the best heuristic in isolation, namely DJT that was introduced by the authors in this publication and which solved 73% of all the problem instances using the optimal number of bins. However the results obtained on subdivisions of the benchmark problems did show promise with, for example, optimal results found for 7 of the 10 so called "hard" problem instances introduced in [104] for which no single heuristic could obtain an optimal solution.

The concept of matching problem state to the best heuristic, used previously in [78, 101], is adopted in [102] where, instead of using a learning classifier system to evolve rules that map problem state to the best heuristic a messy GA [58] is employed

[1] Covered in Section 3.4

for this purpose. The hyper-heuristic uses constructive heuristics, off-line learning and explicit memory in the form of the evolved "rules". The system was tested on a large set of 1016 BPP instances. These consisted of Falkenauer's uniform $(80\,instances)$ and triplet problems (80), Scholl's data sets $1(720)$ and $3(10)$ and 126 newly generated instances.[1] These 1016 problem instances were split into two groups with 763 used for training purposes and 253 used as a test set. Once trained the hyper-heuristic was shown to outperform all of the constituent heuristics on the two sets of problem instances; those used for training and those as yet unseen problem instances in the test set. DJT was shown to be the best individual heuristic in 94.5% of cases when applied to all of the problem instances. The resultant hyper-heuristic was shown to produce solutions of at least a good a quality as DJD for 98% of problem instances and outperformed DJT in 5% of cases.

Terashima-Marin et al.[128] use the same problem characteristics in order to predict the best heuristics for 2 dimensional packing problems. They expand upon previous work [125, 126], using an evolutionary algorithm to evolve general hyper-heuristics that solve regular and irregular two-dimensional bin-packing problems by matching a simplified problem state to the best suited pair of heuristics taken from two sets of simple heuristics, one to select the elements of the problem (bins and items), and another to decide where to place the item. From the wide variety of Cutting Stock Problems (CuSP) the authors limited the study to 2 groups both of which considered only bins, or cutting stock, of uniform dimension but where the items to be cut could either be regular (all rectangular) or irregular (with varied shape). Formally given a set $L = (a_1, a_2, \ldots, a_n)$ of n pieces, each of size $s(a_i) \in (0, A_0]$, to be cut from sheets of size A_0, the objective is to find a packing which minimises the number of cutting stock sheets used. The EA implemented is a *messy* EA with a variable length chromosome, each gene representing both a simplified representation of the problem state and the corresponding pairing of heuristics to be applied. The representation of problem state was different for problems with regular shaped items than for irregular shaped ones but both worked similarly to previous work on one dimensional bin packing problems[101, 102] in which an accuracy based learning classifier system, namely

[1] The problems introduced by Falkenauer and Scholl are decribed in detail in Section 3.3.

XCS, and a messy EA were used to evolve rules in a similar manner. Tested against a set of 1080 problems, taken partly from 4 sets used in the literature with the remainder randomly generated, the approach produced, after being trained on a subset of the benchmark problems, a hyper-heuristic capable of solving unseen instances "very efficiently, in fact, much better than the best single heuristic for each instance."

There are many examples of constructive hyper-heuristics in the literature that have been applied to different problem domains (See [12] for an overview). Although this thesis is restricted to the BPP the overseeing hyper-heuristic approaches used by otheres are still of interest. A few examples are given in the remainder of this section.

Garrido and Riff [53, 54] use a genetic algorithm as an on-line hyper-heuristic for solving 2D strip packing problems. The approach decides which of three categories of underlying heuristics to apply at each step; greedy, ordering or rotational heuristics. The authors report good results that outperform more specialised algorithms for some of the problem instances that the approach was tested on.

Burke et al.,[17] solve educational timetabling problems using a tabu search hyper-heuristic that searches over a pool of 5 constructive graph colouring heuristics by perturbing both the order that the heuristics are applied and the number of events scheduled by each heuristic. Permutations that have been applied previously are kept in a tabu list and excluded as potential moves from the current solution for a number of iterations in order to stop the approach continuously searching over the same portion of the heuristic landscape. The approach was applied to the commonly used Carter benchmark problems [24] and the results contrasted against those obtained using 9 "fine-tuned bespoke state-of-the art approaches" with the hyper-heuristic found to be "capable of generating comparable results to those of special purpose approaches."

Thabtah and Cowling [132] compare the predictive abilities of associative classification techniques to traditional classification methods. The classifiers attempt to forecast which heuristic from a set of underlying heuristics will be most suitable for solving a scheduling problem taken from [30]. The study attempts to data mine, from a set of 16 solutions, the heuristic chosen previously using a "peckish" hyper-heuristic for the problem state. Three associative methods, MCAR[130], MMAC[131] and CBA[75] were contrasted against two conventional techniques, PART[48] and RIPPER[29]. The

results obtained show the associative classification approaches were able to predict the selection of low-level heuristics from the data sets more accurately than conventional classification algorithms.

The Cross-domain Heuristic Search Challenge (CHeSC)

Before reviewing the literature surrounding generative hyper-heuristic techniques it is worth mentioning the first Cross-domain Heuristic Search Challenge (CHeSC) which took place in 2011 with entries from 20 international practitioners. The competition used its own framework named HyFlex [85] which allowed competitors to develop their own selective hyper-heuristics. The hyper-heuristic was responsible for iteratively selecting and applying one of a number of problem specific perturbative heuristics to improve the quality of initial solutions that were instantiated using common domain specific constructive heuristics. The same hyper-heuristic was applied to five problem instances for each of six problem domains, four of which were known and supplied to competitors prior to the competition. The four domains known *a priori* were Max-SAT, 1D Bin Packing, Personnel Scheduling and the Flow Shop problem with the hidden domains, unknown before the competition, being the Vehicle Routing and Travelling Salesman problem domains. The HyFlex framework hid any domain specific information from competitors providing only a measure of solution quality as feedback to their hyper-heuristic. For each domain 4 broad classes of heuristics were provided; local search, mutation, ruin-recreate, and crossover. A competitors hyper-heuristic could work by applying these heuristics to perturb either a single solution or they could operate on a population of solutions. Due to the nature of HyFlex, which incorporated a domain barrier to abstract the user from the different problem domains, competitors were restricted to developing selective hyper-heuristics that searched over a set of perturbative heuristics with only on-line learning allowed. Each competitor's algorithm was applied to a each problem instance from each domain for a set time limit with the final solution quality used to rank approaches. Results were graded using a "Formula One Point System" where for each problem instance the top rated eight algorithms received scores of 10, 8, 6, 5, 4, 3, 2 and 1 point respectively with the remainder of the algorithms scoring 0. The 20 hyper-heuristics entered used a

variety of techniques ranging from evolutionary algorithm to simple hill climbing algorithms. A selective hyper-heuristic was entered to CHeSC during the initial year of this study which used a combination of tabu search and simulated annealing to control the selection and acceptance of heuristics [109]. The algorithm achieved 9^{th} place overall which varied considerably between domains. The algorithm achieved its best result of 4^{th} place on both the Vehicle Routing and Max-SAT problems but dropped to 14^{th} place on the problem instances used for the Flow-Shop and Travelling Salesman domains. This variation in performance across the different domains was also experienced by other competitors entries leading to the observation that hyper-heuristics, as with other search methodologies, are not immune from the requirement to be tuned for specific problems in order to achieve their maximum potential. While the competition served as a good introduction to the field of hyper-heuristics and introduced a number of guiding metaheuristic techniques and problem domains the limitations imposed allowed only a subset of the field of hyper-heuristics to be explored.

Summary of Selective approaches

Reviewing the literature relating to selective hyper-heuristics highlights two common conclusions.

- Selecting a heuristic (or combination of heuristics) from a pool of different heuristics for each problem instance can increase the solution quality when compared to using a single heuristic across a broad range of problem instances
- It is possible to use data mining and classification techniques to predict which will be the best heuristic based upon the characteristics of the problem instance / partial solution.

These observations are investigated in Chapter 4 where an EA is used in an attempt to improve the accuracy obtained using an off the shelf classification technique to predict the best constructive heuristic to apply to each of a large set of benchmark problem instances for the BPP domain.

The remainder of this chapter reviews the literature covering the second broad classification of hyper-heuristics; those that aim to automatically generate novel heuristics from constituent component parts.

2.3.3 Heuristic Generation Techniques

Originally described as "heuristics to select heuristics" the term hyper-heuristics has since been redefined to encompass emerging strategies that are described as "heuristics to generate heuristics" [14]. The majority of approaches that are defined as generative hyper-heuristics utilise Genetic Programming (GP) to construct a procedure, or formula, for solving a problem instance. Genetic programming encompasses many different techniques that are collectively described under the same banner. These include the original Koza [72] style tree structures constructed from mathematical operators and problem specific variables that are used to solve dynamic programming problems to more coarse grained approaches such as Grammatical Evolution which combine grammatical elements that represent programmatic elements [90]. Most of the literature surrounding generative hyper-heuristics, especially in the domain of bin packing, describes conventional Koza style genetic programming techniques to evolve combinations of problem specific variables and arithmetic operators to define a mathematical formula that is used to determine the selection and placement of items.

Packing and cutting problems are amongst the most studied domains in the context of generative hyper-heuristics which have also been designed and applied to a number of other combinatorial optimisation problems including:

- Production Scheduling [36, 56, 61, 123]
- Satisfiability [3, 50, 76]
- Travelling Salesman Problem [69, 88]
- Timetabling and Scheduling [2, 94]

The remainder of this chapter summarises the literature surrounding the field of generative hyper-heuristics concentrating on those approaches that have been used to solve packing and cutting problems.

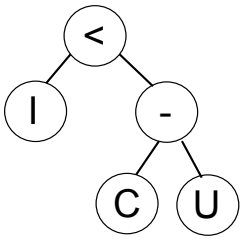


Figure 2.3: First fit heuristic: The tree is evaluated from the top $less - than$ node. If it evaluates as true the item is packed into the current bin otherwise a new bin is opened and it is inserted there. The tree can be expressed by reading left-hand branches first as: *if the length of the item to be packed, I, is $less - than$ the bin capacity, C, minus the used capacity, U, then pack the item into the current bin otherwise pack the item in a new bin.*

The first hyper-heuristic to use Genetic Programming to evolve heuristics for application to the BPP is presented in [9]. The authors use GP to generate constructive heuristics by combining 6 arithmetic function nodes (add, subtract, multiply, protected divide, abs and less than) and 3 problem specific terminal nodes (used bin capacity, total bin capacity and item size). The simple evolved heuristics are shown to be competitive when compared with the performance of "First Fit" averaged over a small set of 20 benchmark problem instances taken from [40]. These problem instances all have a bin capacity, $C = 150$ with each instance comprising of 120 items with weights uniformly distributed in $\omega \in (20, 100)$. It should however be noted that the instances were initially used in the off-line BPP and all have optimal solutions at the lower bound given by $\lceil \sum \omega \div C \rceil$ ranging from 46 to 52 bins. In comparison the FF heuristic requires on average 33% more bins ranging from 57-70 bins when presented with the problem items in the order that they are published. Given the specificity of the small number of nodes used in the study it is maybe to be expected that the heuristics evolved produced solutions of comparable quality to the first fit heuristic. If described in terms of the function nodes used in the study the first fit heuristic can (almost) be expressed as depicted in Figure 2.3. If the less-than function node was replaced with a less-than or equal to node then that heuristic would be identical.

The article does highlight, albeit in a small study, that GP has potential to evolve heuristics that are at least competitive with simple human designed constructive deter-

ministic heuristics.

The system presented in [9] is simplified in [16] by making the observation that the 3 terminal node used previously can be expressed using only two nodes representing the item size and the free space in each bin. Burke et al. compare their evolved heuristics to the best-fit heuristic on newly generated problem instances. Heuristics are first evolved on 7 classes of problem with each of the 7 training sets comprised of 20 newly created problem instances. Each of the 7 evolved heuristics are then evaluated on combinations of another set of 140 unseen problem instances created using the same 7 parameter settings that governed the distribution of item sizes in the corresponding training set. All problem instances were generated with a bin capacity of 150 and comprised of 120 items with weights uniformly sampled between 7 ranges as shown below.

$\omega \in [10, 29]$	$\omega \in [30, 49]$	$\omega \in [50, 69]$	$\omega \in [70, 89]$
$\omega \in [10, 49]$	$\omega \in [50, 89]$	$\omega \in [10, 89]$	

In the experiments described 50 runs are conducted using a population size of 1000 and halted after 50 generations for each of the 7 training problem instance classes. From each run the heuristic with the best average performance on the corresponding training set is retained, resulting in 350 heuristics. Each of the 50 heuristics evolved for each of the 7 classes of problem is then tested on each of the 7 test sets and the results obtained are contrasted with the ability of the Best Fit heuristic. The author show that 14 of the 350 evolved heuristics outperform BF. A further 24 heuristics give worse average performance than BF and the remaining 312 having comparable performance albeit only on the test set that is generated using the same distribution of items as those instances used during training. The main contribution of the paper is in showing how the heuristics that are evolved using a training set that is comprised of problem instances with a wider distribution of item sizes generalise over a wider range of unseen problem instances than those that were evolved using problem instances with more constrained item size distributions. All of the heuristics that outperformed BF on the test set were trained on classes of problem instances with small ranges of

item sizes showing that whilst it is possible to create more specialised heuristics for niche areas of the problem space these heuristics are in general unable to generalise over more diverse problems. Conversely heuristics generated using wider ranges of item size generalise well across larger portions of the problem space but are unable to specialise to niche areas. The authors found this in all but the case where the item weights were in the range 50-69 which they conjecture, is the range that BF is best suited to. It has also been noted previously in the literature that packing problems where the average item size is around a third of the bin capacity are the most difficult . Given the large number of fitness evaluations used in the study in conjunction with the minimal set of problem specific function and terminal nodes it is maybe unsurprising that the heuristics generated maintained similar performance when evaluated on the same class of problem instance from which they were evolved.

The scalability of the technique described above is investigated in [15] where the authors found that the technique scaled well to much larger off-line BPP instances, albeit that those problem instances were generated with item sizes sampled from the same distribution as the previous study. Burke et al. suggested that future studies could be conducted into automatically generating heuristics for the off-line version of the BPP [16] noting that heuristics designed for application to the off-line version of the BPP are typically more complex than those used for the on-line problem. In order to achieve this the authors suggest that this would require the creation of a more complex grammar, able to express the type of human designed heuristics used to solve the off-line problem. This is in part addressed in [22] where the authors use grammatical evolution [90] to generate local search heuristics that are tested on 70 BPP instances taken from the literature. The results presented are comparable with those attained using state of the art metaheuristic search methods on the small number of problem instances that were used in the study. The evolved heuristics are used to select, empty and repack a number of bins taken from a complete solution that is initialised by randomising the order of a problem instance's items and applying the first fit heuristic.

Poli et al. [95] introduce "A Histogram-matching Approach to the Evolution of Bin-packing Strategies" which employs Linear GP to evolve different item selection strategies that are incorporated into a manually designed constructive heuristic. The

authors use the technique to create constructive heuristics which are evaluated on the off-line BPP. The technique is reminiscent of the Sum of Squares heuristic [32] for the on-line BPP which attempts to pack each item so that the number of bins with an equal amount of remaining free space is minimised. In [95] two histograms are maintained that map to the remaining item sizes O and the free space available in used bins G. For a problem instance with bin capacity C there are C values maintained in each histogram i.e. $O = (o_1 \dots o_C)$ and $G = (g_1 \dots g_C)$. The objective of the approach is to pack items until $g_s \geq o_s \forall s \in [1 \dots c]$ (the algorithm continues packing items until the remaining items all fit within the free space available within the set of bins used). At this point the remaining items are simply placed into a bin with corresponding free space. For any item to be packed the strategy selects the first bin with free space into which it will fit in ordered in ascending order of free space. This is the same procedure that the best fit heuristic employs. Results show the evolved heuristics marginally outperform BFD on certain problem classes but achieve worse results on others. The authors conclude that the evolved heuristics are comparable with the best human designed constructive deterministic heuristics. On one class of problem instances where BFD achieves results at the lower bound all but 1 of the 13 evolved heuristics give identical results. This class of problems, with item sizes in the range $[1 \dots 80]$ and bin capacity 150, have been shown to be well suited to solving using BFD. On other classes of problem such as those problem instances with weights in the range $[1 \dots 150]$ with a corresponding bin capacity of 150 none of the 13 evolved heuristics matches the results obtained using BFD. This initial study is constrained by the fact that GP is used to evolve only a small part of the overall hyper-heuristic; the selection strategy, which is incorporated into a manually designed heuristic.

In [11] genetic programming is used to generate constructive heuristics for the two dimensional strip packing problem. Conventional Koza style GP is employed to generate heuristics, represented as tree structures, for selecting and placing pieces. The objective is to minimise the width of the fixed height strip that the rectangular pieces are cut from. The approach considers the problem as a two dimensional bin packing problem in which the width of the bins is changeable. The authors use a variety of terminal nodes that reflect the condition of the partial solution and the pieces still to be

packed.

- The width W of the piece
- The height H of the piece
- Bin Width Left BWL Difference between the bin and piece widths
- Bin Height BH Bin bases height, relative to base of sheet
- Opposite Bin OB Bin height minus height of opposite bin
- Neighbouring Bin NB Bin height minus height of neighbouring bin

The approach is tested on a number of different benchmark problems from the literature. On-line learning is incorporated via an evolutionary process which implicitly matches a problem instance to an evolved heuristic. The results obtained were found to be superior to the best "human-designed state of the art constructive heuristics."

Allen et al. [1] address the three-dimensional knapsack problem using GP to evolve heuristics that are compared to results achieved using best-fit and a simulated annealing method taken from the literature. Their results are worse than best-fit on most of the 200 problem instances that the procedure was tested on but are competitive when compared to the simulated annealing method. The authors hypothesise that the evolved heuristics are only suitable for reapplying to problem instances of the same class to which they are evolved. This work was extended in [20] where one, two and three dimensional bin packing and knapsack problems were tackled using the same representation for all three problems. The authors achieve results comparable with the best human designed heuristics for problems of all dimensions without the need to alter parameters between runs.

Burke et al. [21] improve on previous work [9, 15, 16] by utilising a memory mechanism during the training process to evolve heuristics for the on-line 1D BPP. 20 problem instances containing 500 pieces randomly generated from 4 different size range distributions are used to evolve heuristics that are tested on 240 problem instances with 5000 items each drawn from the same distributions. The memory mechanism is used to record the distribution of item sizes encountered and to alter the strategy

for packing pieces based on this information. They show that the memory enhanced algorithm outperforms the system without memory by a significant margin.

The on-line 1D BPP is tackled in [91] where the authors use "policy matrix evolution for generation of heuristics". The procedure determines the placement of the next piece based on memory retained by a *policy matrix* which is used to maintain a record of the distribution of the available free space in the open bins. Items are packed so as to minmise the number of bins with equal free space. The system is tested on a large set of newly generated problem instances on which the procedure is shown to outperform human designed heuristics.

Generative hyper-heuristic approaches have been used to create heuristics for a range of combinatorial problems including scheduling [66, 123], satisfiability [50] and the travelling salesman problem [88]. Some of these approaches are briefly reviewed in the remainder of this section.

In [76] code is automatically generated in a subset of the programming language ML to solve "Boolean Optimization Problems (BOOP)" using a system called ADATE (Automatic Design of Algorithms through Evolution) [87] which the authors classify as "an off-line, heuristic generating, learning hyper-heuristic." The approach is comparable with genetic programming differing in that the building blocks contain complete tentative programs rather than the lower level building block typically used in genetic programming. The authors use the system to generate replacement move operators for a tabu search metaheuristic using off line learning by training it on a number of problem instances. Experimental results showed the newly generated code produced superior solutions than those attained with the unaltered tabu search algorithm.

Following on from previous work [17], Qu, Burke and McCollum [97, 98] adaptively hybridise graph colouring heuristics in order to generate new heuristics that are applied to benchmark exam timetabling and graph colouring problems with results "competitive with the state of the art human produced methods."

Fukunaga [50] uses genetic programming techniques to generate local search heuristics for the SAT problem and found them competitive to the best human developed heuristics such as GSAT[107], GSAT with random walk [106], Walksat[108] and varients including Novelty[81] and Novelty+[62].

Tay and Ho [123] use genetic programming to evolve "composite dispatching rules" for multi-objective flexible job-shop problems. The results presented show that the framework developed outperforms other common methods on five problem sets.

Hutter et al.[63], develop a parameter optimisation framework (ParamILS) using an iterated local search procedure that they use to tune the parameters in a number of commercial applications including CPLEX, a commercial integer programming solver, with the results obtained showing "substantial and consistent performance improvements". The job of tuning parameters for algorithms is one that has received much attention due to its affect on potential solution quality. Manually tuning an algorithm often constitutes a large proportion of development time due to the laborious and ad-hoc approach typically taken. The paper presented is included in the bibliography of hyper-heuristics maintained by the University of Nottingham available from the CHeSC website [86] along with other parameter tuning approaches to algorithm optimisation. Whilst not obviously falling into the hyper-heuristic definition, the process of automated parameter tuning by applying an algorithm on top of the algorithm to be optimised could be seen as generating a new heuristic, especially in the case where the number of parameters increases. In the case of CPLEX there are "about 80 parameters that affect the solvers search mechanism and can be configured by the user to improve performance".

ParamsILS was used in [71] to optimise parameter settings in the authors SATenstein framework, used to control the design of algorithms for the satisfiability problem,. The SATenstein framework uses 41 parameters to control the recombination of heuristic components taken from 3 well known classes of stochastic local search algorithms which have been broken down into their constituent parts and can be recombined in a total of 4.82×10^{12} different ways. The authors of ParamILS "believe this automated design of algorithms from components will become a mainstream technique in the development of algorithms for hard combinatorial problems" [63].

Summary of Heuristic Generation Techniques

GP offers a partial solution for generating new heuristics for combinatorial problems and therefore takes away the need to design and implement heuristic methods for in-

clusion in selective hyper-heuristic approaches. It fails to be a complete solution as the individual building blocks, or nodes, that are combined using GP still have to be defined for the problem domain being tackled.

The most successful generative hyper-heuristics are perturbative approaches, such as [20]. However, such approaches suffer from many of the problems associated with metaheuristic search; they are computationally expensive[1] and do not provide any assurance of solution quality due to their stochastic nature. They do however offer competitive solutions to human designed heuristics and provide an interesting avenue of research for further study. Future hyper-heuristic research could combine elements of both automatically generated constructive and improvement heuristics with continually adapting selection mechanisms that continue to learn associations between problems and heuristics over time as the nature of the problems presented changes.

2.4 Summary

The term hyper-heuristic is a broad term to describe a number of diverse but related approaches that attempt to address the weaknesses and complexities associated with all problem specific algorithms. Selecting, combining, generating or adapting heuristics for the problem being addressed gives rise to the possibility of generic algorithms able to tackle wider variations of problem instances, even transcending domain barriers. Industry is often reluctant to invest in stochastic metaheuristic methods relying instead on simple, fast deterministic heuristics that have proven capabilities. Hyper-heuristics aim to address many of these concerns by providing generic algorithms able to solve a wide variety of problem instances without the associated expense and skill required to manually tailor metaheuristic search methods for each different class of problem.

The research presented in this thesis addresses a fundamental goal of hyper-heuristics: to select or generate the best heuristic for a particular problem instance. This is addressed directly in Chapter 4 by exploiting a derived mapping between the characteristics of a problem instance and the suitability of a range of individual human designed deterministic heuristics for solving the problem. In subsequent chapters the differences

[1] In [20] a different heuristic is evolved for each problem instance using 50, 000 function evaluations

inherent in all heuristics is further exploited by co-evolving sets of complementary heuristics that individually tailor themselves to niche areas of the problem space whilst collectively collaborating to better cover the much bigger problem space defined by a large corpus of problem instances that were generated using widely varying characteristics.

The key points emerging from this review that have guided the research presented in the remainder of this thesis include:

- Using intelligent combinations of simple heuristics provides better solutions than can be attained by using those heuristics in isolation.
- It is possible to learn a mapping between a problems characteristics and the most suitable heuristic for solving that problem.
- Automating the heuristic design process allows for both generalised and specialised heuristics to be created which can outperform human designed heuristics.
- Different heuristics have different utility on different parts of the problem space giving rise to the possibility of creating sets of heuristics that collectively generalise better over larger problem spaces than any individual heuristic and individually specialise on niche areas of the problem space.

The remainder of this thesis concentrates on addressing the research questions outlined in Section 1.1. For a hyper-heuristic to be effective the hypothesis is that the set of heuristics that it is searching over should have different utility on different portions of the search space thereby providing the hyper-heuristic with the ability to improve solution quality by selecting the most appropriate heuristic for each problem instance presented. The following chapter investigates the performance of a number of deterministic constructive heuristics applied to a large set of benchmark problem instances and analyses the performance of these heuristics in relation to a number of problem characteristics and also to each other.

Chapter 3

Benchmark Heuristics and Problem Instances

3.1 Introduction

The remainder of this thesis focuses on hyper-heuristics specifically selective and generative hyper-heuristics that utilise deterministic constructive heuristics for the off-line BPP. The focus of this chapter is to examine a selection of these simple heuristics and analyse their performance when applied to a large set of commonly used benchmark problem instances sourced from the literature. The analysis focuses on investigating potential relationships that exists between a problem instance's characteristics and the quality of the solution produced by each heuristic. The relationship between problem and difficulty, from the perceptive of individual heuristics, is presented in relation to a number of problem characteristics. The term difficulty is used throughout this chapter to describe problem instances from the perspective of individual heuristics. The chapter attempts to find general correlations between problem characteristics and problem complexity but as is more often the case problem difficulty has to be measured with respect to the method used to solve the problem.

3.1.1 Contribution

As well as a detailed investigation of a wide range of benchmark problem instances and deterministic heuristics for the BPP the chapter introduces a new deterministic constructive heuristic (ADJD) which is shown to provide solutions of higher quality on problem instances with smaller item sizes than those produced by the other heuristics investigated. Two newly generated large sets of problem instances are also presented that and are shown to be more difficult for the heuristics investigated than other problem instances commonly used in the literature. The chapter provides evidence of the potential utility of hyper-heuristic approaches that utilise diverse sets of simple heuristics with differing capabilities when contrasted to the quality of solutions attained using a single heuristics evaluated over a broad range of problem instances with widely varied characteristics.

3.1.2 Background and Motivation

Many practitioners who introduce new methods for solving the BPP evaluate their approaches against a *subset* of widely used benchmark problem instances that were introduced in two publications [40, 104]. Whilst benchmark problem instances are a useful indicator, that allow for practitioners to evaluate their results against established methods, often the choice of problem instances and the simple metrics used can distort findings [4, 38]. Highly specialised algorithms such as Falkenauer's Hybrid Grouping Genetic Algorithm (HGGA) give good results when evaluated on problem sets that they were designed to solve, or in the case of Falkenauer's HGGA that were generated concurrently and released in same publication [40] .

It is well known that any individual heuristic is unable to perform well across all possible problem instances in a domain [135] . For the on-line version of the BPP where item sizes are not known a-priori it has been shown the no heuristic has better average and worst case performance than the best-fit heuristic over all possible problems [70]. These mathematical theorems, whilst making important points are based on a theoretical set of infinite problem instances. On small sets of problem instances, generated within finite parameter ranges, it is possible to design or even automate the

creation of heuristics that outperform best-fit [16].

This chapter explores the relationship between the performance of a selection of deterministic heuristics and the characteristics of a large number of benchmark problem instances widely cited in the literature, with the intention of highlighting key problem characteristics that affect a heuristic's performance. This knowledge is exploited in subsequent chapters to facilitate the creation of both selective and generative hyper-heuristics that use or create diverse sets of heuristics thereby increasing the potential for those hyper-heuristics.

The *algorithm selection problem* was first introduced in the seminal work of Rice [99] and was largely investigated separately by the machine learning and metaheuristic communities over the subsequent 2 decades. Smith-Miles [119] proposes a cross disciplinary framework which tries to address the algorithm selection problem by incorporating machine learning techniques into non learning algorithms. Investigations into the selection of appropriate metaheuristic techniques in the domains of constraint satisfaction and graph colouring are conducted in [117] and [118] respectively. The problem as defined by Rice is illustrated in Figure 3.1. Smith-Miles proposes the incorporation of machine-learning techniques into Rice's model to facilitate learning a mapping between the feature space of a problem and the algorithm space defined by a number of metaheuristics for solving a range of cross-disciplinary problems. The work presented in this chapter concentrates on the Feature Space F of a large set of problems from the BPP domain and attempts to ascertain if any obvious correlations exist which could provide good indicators of algorithm performance.

The remainder of this chapter is structure as follows. Section 3.2 describes three metrics used to evaluate the quality of the solutions produced for a BPP instance. Section 3.3 describes some of the most widely used benchmark problem instances taken from the literature for the off-line BPP and introduces two newly generated large problem sets totalling almost 20,000 problem instances. Section 3.4 describes a range of deterministic heuristics frequently used in the literature and presents a new novel deterministic heuristic, *Adaptive Djang and Finch* (ADJD), introduced to address weaknesses inherent in the other heuristics reviewed when applied to problem instances with certain characteristics. The chapter concludes with an analysis of these benchmarks,

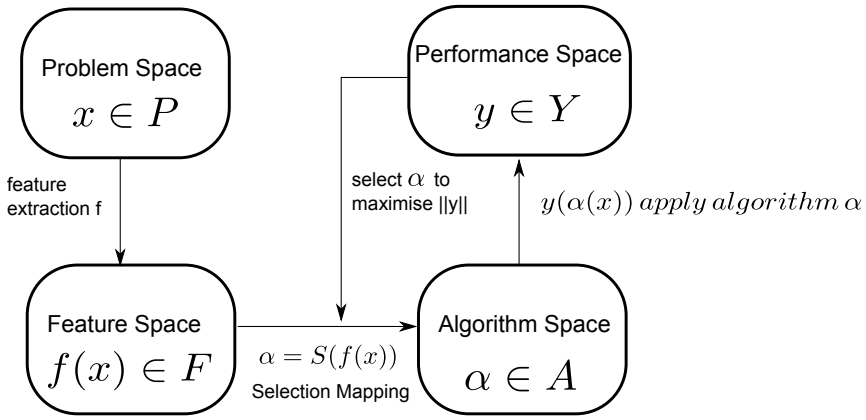


Figure 3.1: Rice described the algorithm selection problem as the task of finding a mapping between the feature space F (characteristics) of a set problems P to algorithm space defined by the set of available algorithms A such that the performance as measured by one or more metrics Y is optimised

exploring the relationship between the characteristics of a problem instances and the effectiveness of the heuristics.

3.2 Metrics

Before introducing the problem instances and deterministic heuristics used throughout this thesis for comparison, three metrics that are used to evaluate the solutions obtained are described. Many publications present results obtained on the BPP by simply indicating the number of instances solved using the known optimal number of bins, given by Equation 2.1, or if the optimal is unknown, the fewest number of bins reported in the literature. This binary metric of success gives no indication of suboptimal solution quality and does not allow differentiation between different solutions to the same problem instance that use the same number of bins.

Hyper-heuristics aim to provide procedures that obtain acceptable quality solutions to a wide variety of problem instances rather than being tailored to a specific class of problem and therefore the quality of sub-optimal solutions is also of interest. Consequently as well as the number of optimal solutions obtained, two other metrics are used in the remainder of this thesis in order to allow a more precise indication of a

heuristic's performance. The first metric used that gives a finer measure of suboptimal solution quality than the binary measure of optimality is simply the number (or ratio) of extra bins required over and above the optimal number. The second and highest precision metric used here was introduced by Falkenauer and Delchambre [41] and is given in Equation 3.1. This equation is used throughout this thesis with $k = 2$ as it was originally presented in the literature. The equation allows, with $k > 1$, a distinction to be made between different solutions to the same problem instance that use the same number of bins. Solutions are rewarded if any free bin capacity is restricted to as few bins as possible. This seems a sensible choice as it rewards heuristics that fill bins *early on* and although this study is restricted to the off-line BPP many of the heuristics introduced can be used without modification for the on-line BPP where the ability to fill bins completely and quickly is of more concern.

$$f(x) = \sum_{j=1}^{n} \left(\frac{fill_j}{c} \right)^k \div n \tag{3.1}$$

This metric provide a means of distinguishing between the quality of different solutions to the same problem instance, for both optimal and suboptimal solutions, that use the same number of bins.

3.3 Benchmark Problem Instances for the BPP

Using standard problem instances allows new algorithms to be contrasted with previously documented techniques. Benchmarks could be described as being unrealistic compared to real world problems, as lacking enough diversity[1] to allow an effective comparison of techniques for a given domain and even encouraging bad practice by limiting the scope and effectiveness of search techniques. Much of the literature in the BPP domain over the past decades has concentrated on the development of more and more complicated search techniques tailored to solving small subsets of a limited

[1] Diversity is used here in relation to the parameters that the problem instances were generated from. Many problem instances are generated by producing a solution at the lower bound using item weights uniformly distributed over a narrow range. In contrast the optimal number of bins is rarely known for real world problems.

number of benchmark problem instances. Practitioners often highlight the benefits of their approaches over others by publishing minimal increases in performance on unrepresentative niche sets of problem instance.

A fundamental goal of hyper-heuristic research is to develop more general techniques that produce satisfactory solutions within an acceptable time scale for a large, diverse range of problem instances. Selecting or generating heuristics that perform well on diverse sets of problem instances is therefore a key concept behind the research presented in this thesis. In order to evaluate any approach benchmark problem instances remain an important tool. Subsequent chapters present a number of techniques that are evaluated on the most commonly used benchmark problem instances from the literature (described here) as well as two large sets of newly generated problem instance introduced in Section 3.3.7. Unlike much of the literature the hyper-heuristics developed during the course of this research are evaluated on multiple complete benchmark problem sets rather than "cherry picking" those instances that are best suited to each technique. The remainder of this section describes a number of historical studies from which the most commonly cited benchmark problem instances used in this thesis have emerged.

The following terms are used during the remainder of this and subsequent chapters.

- C is the capacity of each bin.
- n is the number of items in the problem instance.
- ω_i is the weight or size of item $i \; \forall \; i \; \in [1, n]$

3.3.1 Martello and Toth Benchmark Problems

To test the effectiveness of their MTP algorithm Martello and Toth [79] generated 900 problem instances, 20 for each of the 45 combinations of the following parameters.

- $C \in \{100, 120, 150\}$
- $n \in \{50, 100, 200, 500, 1000\}$
- $\omega \in [1, 100], [20, 100], [50, 100]$

Item weights (ω) were generated randomly from a uniform distribution for the given ranges.

3.3.2 Falkenauer Benchmark Problems A

To evaluate his Genetic Grouping Algorithm Falkenauer [43] created problem instances using the following procedure. First items were generated that completely filled the capacity of the optimal number of bins. From this initial point the weight of randomly chosen items was reduced so that the sum of all weights was a predefined quantity, ($\alpha\%$) less than than the capacity of one bin. Weights were generated to ensure that bins in an optimal solution contained only one item of less than 25% of the capacity of a bin in order to make the problem *more difficult than simpler uniformly generated distributions* Fifty problem instances were generated for each value of $\alpha \in [1.5, 3, 4.5, \ldots, 15]$ with the bin capacity fixed at $C = 250$ and the number of items generated remaining constant at $n = 64$ giving rise to a test bed of 500 problem instances

3.3.3 Falkenauer Benchmark Problems B

Falkenauer [42] generates two classes of problem. The first, used to compare his Hybrid Grouping Genetic Algorithm (HGGA) to Martello & Toth's Reduction Procedure (MTP) is generated using the same method as described in Section 3.3.2, with the following parameter settings.

- $C = 150$
- $n \in \{120, 250, 500, 1000\}$
- $\omega \in [20, 100]$

20 instances for each parameter combination are generated giving a total of 80 problem instances. These are termed Falkenauer's uniform problems and are given the abbreviation here of $FalU$.

Falkenauer conducts a second experiment to determine the "practical limits" of the HGGA by generating "triplets ... the most difficult BPP instances". For his triplet

class of problems (abbreviated to $FalT$ here) the problem instance item weights are real numbers taken from the range $\omega \in [0.25, 0.5]$ with C fixed at 1. Each instance is generated so that the optimal solution has each bin filled to capacity with no free space in any of the bins. This is achieved by selecting the first item for each bin from the range $\omega \in [0.38, 0.49]$ leaving the free space, s, left in a bin within the range $[0.51, 0.62]$. The second item is then generated from the range $\omega \in [0.25, \frac{s}{2}]$ with the third item selected to make up the remaining bin capacity. This procedure ensures that only one item falls in the range $\frac{C}{2} < \omega < \frac{2C}{3}$ for each bin. No two items generated from this range can be placed in the same bin without exceeding the capacity. The remaining two items are generated such that $\frac{C}{4} < \omega < \frac{C}{3}$. It is clear that if the optimal solution is to be found one item from the set of larger items and two from the smaller set must occupy each bin. Although it may be possible to place three items from the set of smaller items into a bin this would lead to a valid but sub optimal solution. Twenty problem instances were generated for each value of $n = [60, 120, 249, 501]$ giving rise to a second set of 80 problem instances. These problem instances are provided via the OR library [7]. The item weights are supplied with as real values with a precision of 2 decimal places. These are scaled here by a factor of 100 to produce item weights with integer values so as to be consistent with the other problem instances sourced from the literature.

3.3.4 Schwerin and Wäscher Benchmark Problems

Schwerin and Wäscher [105] in their paper "The Bin-Packing Problem: A Problem Generator and Some Numerical Experiments with FFD Packing and MTP" attempt to "identify classes of problem instances which are difficult to solve and therefore can be considered as benchmarks for newly developed methods". The paper criticises the test bed used in [79] and described here in Section 3.3.1 suggesting a different method for generation of problems. Two new parameters are introduced in an attempt to characterise the interval from which the item lengths are obtained in relation to the capacity of the bin. The generators described in 3.3.1 and 3.3.3 produce problems that are characterised by the tuple (C, n, ω) where $\omega = \omega_1 \ldots \omega_n$ denotes the vector of item weights.

Schwerin and Wäscher introduce two new parameters v_1 and v_2. Weights are sampled from the range $\omega \in [v_1C, v_2C] : 0 < v_1 < v_2 \leq 1$ and are selected using the formula $\omega = \lfloor (v_1 + (v_2 - v_1)rand(0,1))C + rand(0,1) \rfloor$

These new parameters allow for the relationship between the lower and upper bounds imposed on the item weights to be defined in terms of the bin capacity something that as noted in [42] can have a profound effect on the "hardness" of a problem instance. One hundred problem instances are generated for each permutation of the following parameters giving rise to a total of 44000 problem instances.

- $C = 1000$
- $n = [20, 40 \ldots 180, 200]$
- $v_1 = [0.001, 0.005, 0.15, 0.25, 0.35]$
- $v_2 = [0.1, 0.2 \ldots, 0.9, 1.0]$

3.3.5 Scholl et al. Benchmark Problems

To evaluate the Bin Packing Solution Procedure (BISON) Scholl et al. [104] generated three data sets. The first set was created similarly to those described in Section 3.3.1 by Martello and Toth but using the following parameters.

- $C = [100, 120, 150]$
- $n = [50, 100, 200, 500]$
- $\omega \in [1, 100], [20, 100], [30, 100]$

For each of the 36 combinations of the above parameters 20 problem instances are generated with weights selected randomly from a uniform distribution between the ranges shown giving a total of 720 instances. This data set is termed $ds1$ here. The second data set, named $ds2$, introduces new parameters in order to specify the average number of items per bin in an optimal solution. The parameters used are as follows.

- $C = 1000$

- $n = [50, 100, 200, 500]$
- $\varpi = [\frac{C}{3}, \frac{C}{5}, \frac{C}{7}, \frac{C}{9}]$
- $\delta = [0.2, 0.5, 0.9]$

The term ϖ specifies the average weight of the items in a problem instance, allowing for either 3, 5, 7 or 9 items per bin on average in an optimal solution. δ is the deviation applied to the average weight. For example if $\delta = 0.2$ and $\varpi = \frac{C}{5}$ the average weight is 200 and the maximum deviation from this value is 40 giving $\omega \in [160, 240]$. For each of the 48 parameter combinations 10 instances were generated with weights selected at random from a uniform distribution giving a total of 480 problem instances. The third data set, named $ds3$ here, uses the parameters $n = 200, C = 100000, \omega \in [20000, 35000]$ in order to create problem instances that are less likely to have duplicate item weights, a feature which the authors state makes problem instances harder by reducing the number of plateaus in the search space caused by duplicate item weights. Ten "hard" problem instances were generated using these parameters.

3.3.6 Summary of Reviewed Benchmark Problem Instances

The problem instances designed by Martello and Toth [79] to evaluate their exact MTP branch and bound algorithm and widely used as a benchmark for future studies [41, 42, 43] are criticised by Schwerin and Wäscher [105] as being biased towards the own authors algorithm. Gent [57] asserts that for the set of problems introduced by Falkenauer and Delchambre [41], their conclusion that 5 of the unsolved problems are "hard" is unjustified. Gent shows that 4 of these problems were easily solvable either by hand or by using simple heuristic methods.

All of the instances from the two datasets $FalU$ and $FalT$, introduced by Falkenauer in [40], have optimal solutions at the lower bound given by Equation 2.1 except for one [57]. Each "class" of problem from $FalU$ varies only in the number of items included taken from $n \in \{120, 250, 500, 1000\}$. Those in $FalT$ vary only in $n \in \{60, 120, 249, 501\}$ and are created in a way to ensure all bins in an optimal so-

lution have exactly 3 items in them and that all bins are filled to capacity. Optimal solutions to all instances from $FalU$ have between 2 and 3 items in each bin.

Falkenauer's HGGA, introduced alongside these benchmark problem instances, uses a novel genome representation and implements both custom crossover and mutation operators all of which are tailored to the BPP domain. Key to HGGA's success however is the hybridisation mechanism; a local search heuristic inspired by [79] which searches for combinations of up to 3 items that optimally filled a bin. All of the 160 problem instances introduced in [40] have optimal solutions with between 2 - 3 items per bin. It is clear that the combination of heuristic and benchmark problem instances used are ideally matched and are not representative of the potentially "infinite" set of possible bin packing problem instances. The results presented for the corresponding algorithm are biased to the problem instances that they are specialised towards solving.

Previous hyper-heuristic approaches that evaluate their methods on subsets of benchmark problem instances are also evident in the literature. Burke et al. [9] use only 20 problem instances from $FalU$, all generated with the same parameters, to evaluate their generative hyper-heuristic. Ross et al. [102] conduct a more thorough study using 890 problem instances comprising of $FalU, FalT, ds1$ and $ds3$ but omit the problem instances from $ds2$. The authors show that DJT, introduced in that paper, performs best when evaluated on all of the problem instances used. When DJT is applied to the problem instances in $ds2$ it performs badly when contrasted with the other heuristics used in the study. Whether practitioners design their algorithms for particular classes of problem, evaluate their algorithms on those classes they perform best on or choose to ignore poor results on unsuitable problem instances is unclear but a fundamental goal of hyper-heuristics is not to generate specialised algorithms that are tailored towards small subsets of all possible problem instances from a particular problem domain but to develop simple procedures that are capable of providing good quality solutions over many diverse problem instances or even across different problem domains.

Of the problem instances described in this chapter those introduced by Falkenauer [40] and Scholl et al. [104] are the most commonly cited in the literature which in conjunction with their availability and variety makes them ideal as a large diverse

collection of problem instances for evaluating hyper-heuristic algorithms. As stated previously all of the 160 problem instances from the two datasets $FalU$ and $FalT$, introduced by Falkenauer in [40], have optimal solutions at the lower bound given by Equation 2.1 except for one [57]. All of the 1210 problem instances included in data sets $ds1, ds2$ & $ds3$, introduced by Scholl et al., in [104] have optimal solutions that may vary from the lower bound given by Equation 2.1. However all optimal solutions are known and have been solved since their introduction [105]. Table 3.1 summarises the parameters from which the benchmark data sets taken from the literature were generated.

Those introduced by Falkenauer are maintained by the OR library accessible at [7] The 3 data sets introduced in [104] are accessible at [103]. Both of these problem sets and other benchmark problem instances for the BPP are mirrored by The EURO Special Interest Group on Cutting and Packing (EPICUP) [89].

Table 3.1: Data sets $ds1$, $ds3$ and $FalU$ were created by generating n items with weights randomly sampled from a uniform distribution between the bounds given by ω. Those in $FalT$ were generated in a way[40] so that the optimal solution has exactly 3 items in each bin with no free space. Scholl's $ds2$ was created by randomly generating weights from a uniform distribution in the range given by $\varpi \pm \delta$. The final column gives the number of instances generated for each parameter combination.

Data Set	capacity (c)	n	ω	#Problems
$ds1$	100,120,150	50,100,200,500	[1,100],[20,100],[30,100]	$36 \times 20 = 720$
$ds3$	100000	200	[20000,30000]	10
$FalU$	150	120,250,500,1000	[20,100]	$4 \times 20 = 80$
$FalT$	1	60,120,249,501	[0.25,0.5]	$4 \times 20 = 80$

Data Set	c	n	ϖ (avg weight)	$\delta(\%)$	# Problems
$ds2$	1000	50,100,200,500	$\frac{c}{3}, \frac{c}{5}, \frac{c}{7}, \frac{c}{9}$	20,50,90	$48 \times 10 = 480$

These 1370 problem instances are referred to as Problem Set A throughout the remainder of this thesis in order to differentiate them from Problem Sets B and C introduced in the following section.

3.3.7 New Problem Instances for the BPP

In subsequent chapters two newly introduced problem sets are utilised along with those in Problem Set A, described previously and summarised in Table 3.1.

- Problem Set B, consisting of 3968 problem instances.
- Problem Set C, consisting of 15830 problem instances.

The problem instances were generated using a custom designed generator that attempts to create problem instances from parameters derived by sampling the problem instances in Problem Set A. The choice of parameters to use was arrived at following the investigation presented in the previous chapter where the affect that different parameters exert on the perceived difficulty of a problem instance was conducted.

Problem instances are generated using the following characteristics.

- The bin capacity C
- The number of items n
- The ratio of small items of size $\omega \leq \frac{C}{4}$
- The ratio of medium items of size $\frac{C}{4} < \omega \leq \frac{C}{3}$
- The ratio of large items of size $\frac{C}{3} < \omega \leq \frac{C}{2}$
- The ratio of huge items of size $\omega > \frac{C}{2}$
- The total free space in the optimal solution summed across all the bins (attempted to be fixed here to 0).

The four ranges of item weights with respect to bin capacity that are used here, small, medium large and huge, were derived from [101] where the authors noted the relevance of these characteristics in relation to the difficulty of a problem instance. All instances generated here have known optimal solutions at the lower bound given by Equation 2.1. Each of the 1370 problem instances taken from the literature [40, 104], summarised in Table 3.1, was sampled in turn to determine the number of small,

medium, large and huge items along with the number of items and the bin capacity. These settings are used by the generator to generate new instances. The generator attempts to create problem instances where the free space summed across all bins is zero, a setting that increases the difficulty of the problem instances. The correct number of items are generated at random from a uniform distribution for each of the four size ranges. Items are then packed into the correct number of bins placing each into the bin with the smallest used space with no restriction imposed on the bin capacity. This results in a solution with bins either over or under packed. Items are then adjusted in size so as to exactly fill the bin capacity C. These items are randomly selected using a roulette wheel selection which gives preference to items in the "ranges" with greater numbers of items. Note that the process is not always successful in producing problem instances with a known solution at the lower bound and these problem instances are discarded. The procedure can result in some instances with a disproportionate number of items with weights at each of the size range boundaries. The problem instances are generated so as to have no free space in the optimal solution. This is also not always successful. However if the solution lies within the lower bounds (the total free space is less than the size of one bin) the problem instances are retained.

For Problem set B, 3 problem instances were generated from each set of parameters obtained from each of the 1370 benchmark problem instances giving a total of 4110 problems. Invalid problem instances were discarded resulting in a set of 3968 problem instances.

For Problem Set C, consisting of 15,830 problem instances, many more valid instances were initially generated. Each problem instance was then solved using 4 deterministic heuristics (FFD, DJD, DJT and ADJD, described in the following section) as well as an implementation of Falkenauer's Hybrid Grouping Genetic Algorithm. Any problem instances for which an optimal solution was found were discarded, resulting in a set of 15,830 "hard" problem instances. These two sets of problems can be downloaded at [111] as SQLITE database files that also provide one example optimal solution for each instance.

The following section introduces a range of deterministic constructive heuristics for the off-line 1D BPP sourced from the literature along with a new heuristic (ADJD)

that was developed during the course of study and presented initially in [115].

3.4 Benchmark Deterministic Heuristics for the BPP

This section describes some of the commonly used deterministic heuristics that have appeared in the literature over the past decades that are used widely to solve instances of the BPP due to their simplicity and their ability to provide quick acceptable quality solutions to these computationally intractable problems. A new heuristic is presented, named Adaptive Djang and Finch (ADJD) that was implemented to address the poor performance of the other heuristics on certain problem instances. The section concludes with an analysis of the performance of some of the most commonly used deterministic heuristics on the benchmark problem instances described previously. The heuristics described below are all designed for application to the off-line version of the BPP where the item sizes are known *a priori*. Each heuristics pre-sorts items by descending order of weight.

3.4.1 Best Fit Descending (BFD)

Best Fit Descending (BFD) [28] puts each piece in turn into the fullest bin that has room for it. If no open bin has sufficient free capacity to accommodate an item then a new bin is opened. All bins remain open for the duration of the procedure. The on-line version of BFD named Best Fit (BF) has been shown to have best possible average and worst case performance over all possible problems [70]. However BFD is identical in terms of the solution quality attained for all but 1 of the 1370 problems in Problem Set A when compared to FFD.

3.4.2 Worst Fit Descending (WFD)

Worst Fit Descending (WFD) [28] places each piece, taken in decreasing order of size, into the bin with the most free space that has room for it. A new bin is opened when a piece does not fit into any existing bin. All bins remain available for the duration of the procedure.

3.4.3 First Fit Descending (FFD)

First Fit Descending (FFD) [28] takes each item in decreasing order of size and places it into the first bin found that will accommodate it. Bins are traversed in the same order as they are opened. A new bin is opened only when the item to be packed does not fit into any of the already open bins. All bins remain open whilst there are still items remaining to be packed.

3.4.4 Djang & Finch (DJD)

Djang & Finch (DJD) [37] pre-sorts all items in descending order of weight before using the following procedure to pack one bin at a time. Items are selected in order and packed into a bin until the bin is filled to at least $\frac{1}{3}rd$ of the bin's capacity. The algorithm then attempts to find sets of items of maximum cardinality 3 that leave a free capacity of zero. If this is not achievable then the constraint is relaxed and sets that fill the bin to 1 less than the maximum capacity are considered. This relaxation repeats until a set of items is found or all permutations have been exhausted. If there are multiple permutations that provide the same level of packing then the one that uses the largest items is preferred. After this partial search the bin is closed and the procedure is repeated with a new bin.

3.4.5 Djang & Finch More Tuples (DJT)

Djang & Finch More Tuples (DJT) [101] DJT works identically to DJD with the exception that it considers combinations of up to 5 items once the initial filling stage is complete and the bin is more than $\frac{1}{3}rd$ full.

3.4.6 Sum of squares (SS)

Sum of Squares (SS) [32, 33] is an on-line bin packing heuristic which puts items into bins such as to minimise the number of bins with equal free space. Sum of Squares is mentioned here as it was used in a paper produced during the course of this study [113] after comments from a reviewer suggesting its use. However the algorithm was

designed as an on-line BPP algorithm and was found to perform poorly when compared to the off-line algorithms described here whether the items were presented in descending, or random order. It should be noted that SS solves all but one of the 80 problems from Falkenauer's $FalT$ problem set if presented with the items in the order that they are published highlighting that the published order effectively defines the optimal solutions. If the problem instances from $FalU$ are tackled in either descending or random order SS produces solutions of worse quality than all of the other heuristics included here.

3.4.7 Adaptive Djang & Finch (ADJT)

Adaptive DJD (ADJD), introduced in a publication resulting from the research presented in this thesis [115], packs items into a bin in descending order until the *free space* in the bin is less than or equal to three times the average size of the items remaining to be packed. It then operates like DJD looking for the set of up to three items that best fills the remaining capacity with preference given to permutations that use the largest items.

ADJD was implemented to address the weakness of the other heuristics on problem instances with smaller item sizes in relation to the bin capacity such as those from $ds2$. Table 3.2 shows that although ADJD is the worst performer on the complete set of 1370 problem instances and is particularly poor when applied to the problem instances from $ds1$, it achieves significantly better results on the problem instances from $ds2$ than any of the other heuristics. The success of ADJD on the problem instances from $ds2$ is achieved by first packing items in descending order of size until the *free space* in the bin is less than or equal to average size of the items remaining to be packed thus improving the chance of finding a combination of up to three items to fill the remaining capacity when applied to problem instances with a smaller average item weight than can be successfully be tackled by either DJD or DJT. Of the heuristics examined here it is the only one that employs an adaptive strategy which alters the behaviour during the course of solving a problem instance and is shown to produce solutions that are noticeably different to those obtained using the other heuristics.

3.4.8 Summary of Deterministic Constructive Heuristics for the off-line Bin Packing Problem

Examining the solutions produced by each of the above heuristics on the 1370 problem instances in Problem Set A highlighted that the solutions produced for many problem instances are similar if not identical even when evaluated using Falkenauer's fitness metric given in Equation 3.1. As an example, Figures 3.2 and 3.3 show visualisations of the solutions obtained using 4 of the heuristics outlined above on 2 different benchmark problem instances, one from $ds1$ and the other from $ds2$. The problem instance names come from the original publication [104]. Colours reflect the item sizes as a ratio of the largest item from red-large to blue-small.

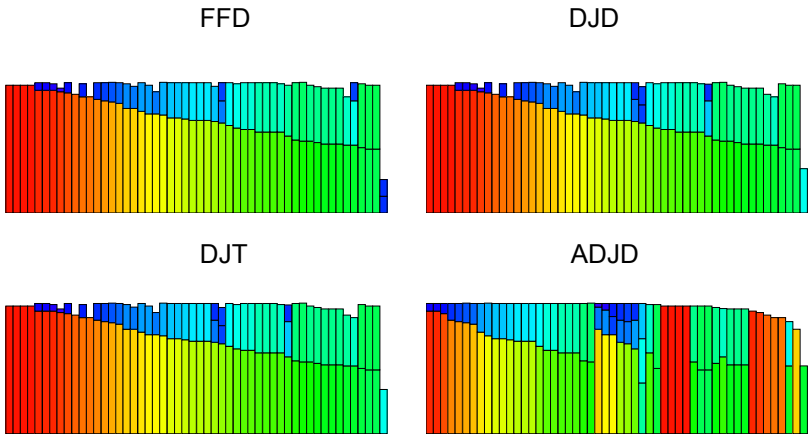


Figure 3.2: The four diagrams depict the solution obtained to problem N2C1W1H taken from Problem Set A, ds1. The solutions produced by FFD, DJD and DJT are almost identical (DJD and DJT's solutions are identical. In contrast ADJD produces a more unique solution when contrasted to those produced by the other heuristics. All solutions are optimal using 52 bins. The solution produced by FFD is ranked highest using Falkenauer's fitness function with the solution produced by ADJD ranking last.

In order to minimise the set of heuristics used for comparison with the hyper-heuristic approaches detailed in the remainder of this thesis, those that were observed to be similar in terms of performance were excluded. The heuristics chosen for providing the most diverse solutions across the problem instances in Problem Set A were FFD, DJD, DJT and ADJD. The remaining sections of this chapter provide an analysis

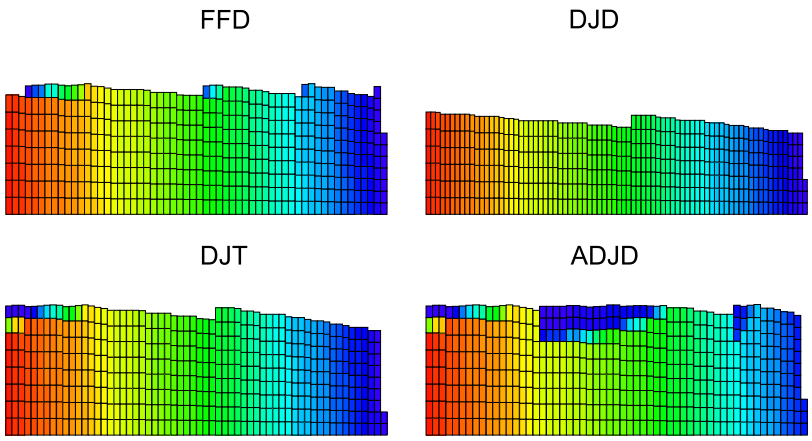


Figure 3.3: The 4 diagrams show the solutions obtained by each heuristic to Problem N4W4B1R5 taken from Problem Set A, ds2. On problems with smaller items DJT finds better solutions than DJD due to its more expensive partial complete search. DJT is still limited when the item sizes are very small. ADJD conversely adapts to these type of problems well and as is shown generates solutions where the items are not as clustered by size as much as the solutions obtained using the other heuristics. The solutions produced by FFD, DJD, DJT and ADJD use 58, 78, 60 and 57 bins respectively. The optimal is 56.

of the performance of these 4 deterministic heuristics in relation to the characteristics that the benchmark problem instances were generated from, outlined in Table 3.1.

3.5 An Analysis of the Performance of Deterministic Heuristics on Benchmark Problem Instances Relating to Problem Characteristics

It has been noted [57] that many so called "hard" benchmark problem instances can be solved easily by simple procedures. Often benchmark instances are introduced in the literature alongside procedures specifically designed to solve them, such as those from Falkenauer [42] whose Hybrid Grouping Genetic Algorithm (HGGA) utilises a local search heuristic inspired by Martello and Toth's Reduction Procedure (MTRP) [80] tailored for finding optimal sets of three items. It has been shown for FFD and

MTRP [105], and thus DJD and HGGA which both use searches inspired by MTRP, that instances with average weights, $\varpi_j \rightarrow \frac{c}{3}$ are the most difficult with those where $\varpi_j \rightarrow \frac{c}{4}, \frac{c}{5}, \frac{c}{6} \ldots$ proving difficult also[1]. All of the problems used here, except for those in ds2, have an average item weight of approximately $\frac{c}{3}$.

In [101] the authors showed DJT to be the most successful heuristic when used in isolation solving 73% of instances to the known optimum. The study used all of Problem Set A, defined earlier in Table 3.1 with the exception of $ds2$, on which DJT finds only 45% of the optimal solutions[2]. The developers of DJT extend their heuristic to counter this weakness by using a *"filler"* method which continues to place single items into the bin after the largest set of five items is placed. This is repeated until no more items can be found. However if this filler process is invoked on any bin then it follows that the search process was unable to find any combination of 5 items due to the small average weight of the problem instance's items and that any positive effect that was intended to emerge from the computationally expensive search procedure is lost. ADJD, introduced here, whilst the worst performer on the complete set of problems from Problem Set A achieves significantly better results on the problem instances from $ds2$. This is accomplished by first repeatedly packing items in descending order of size until the *free space* in the bin is reduced to less than or equal to average size of the items remaining to be packed. This improves the possibility of finding a permutation of up to 3 items that will fill the remaining capacity for problems with smaller average item weights than can be successfully tackled using either DJD or DJT. The remainder of this section investigates the relationship between problem difficulty and the characteristics of the problem from the viewpoint of 4 benchmark heuristics. This is primarily conducted on the problem instances from problem set A as the other two problem sets were generated towards the end of the period of study culminating in this thesis.

[1] If a solution exists at the lower bound given in Equation 2.1 then the total free space $\varpi_{free} \rightarrow 0 \; as \; \varpi_j \rightarrow \frac{c}{i} : i \in \mathbb{N} : i \geq 3$

[2] DJT will perform best where $\varpi \geq \frac{2}{15}c$ as once the initial filling procedure has filled $\frac{1}{3}c$ the remaining $\frac{2}{3}c$ can be filled by at most five items.

Table 3.2: Benchmark heuristics performance on the benchmark problem instances

	ds1 720 Problems		ds2 480 Problems		ds3 10 Problems		falk U 80 Problems		falk T 80 Problems		All 1370 Problems	
	solved	bins	solved	bins	solved	bins	solved	bins	solved	bins	solved	bins
FFD	75.83	0.36	49.17	3.69	0	6.05	7.5	1.3	0	14.21	57.52	1.78
DJD	79.03	0.28	21.04	9.77	0	3.56	57.5	0.27	0	2.45	52.26	2.00
DJT	83.75	0.17	44.58	2.66	0	3.56	57.5	0.27	0	2.45	62.99	0.73
ADJD	35.83	1.32	80.21	0.66	0	5.87	53.75	0.33	0	1.68	50.07	1.12

Table 3.3: Extra bins (δ) required by 4 deterministic heuristics compared to the best known solutions from the literature on the 1370 benchmark problem instances in Problem Set A.

	Number of Problems Solved Requiring δ Extra Bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
FFD	788	267	78	83	39	16	18	9	18	4	50
DJD	716	281	119	58	48	36	10	16	23	3	60
DJT	**863**	331	90	26	30	15	11	2	1	1	0
ADJD	686	368	153	76	38	22	12	9	1	5	0

3.5.1 Solution Quality

Table 3.2 summarises the results obtained, for each of the four deterministic heuristics used throughout this thesis on the set of 1370 problem instances outlined in this chapter grouped by the data sets (problem characteristics) as they were published. Both the percentage of problems solved optimally and the average percentage of extra bins required are given. It is interesting to note for instance, that whilst FFD rates highly if ranked in terms of the number of optimal solutions found, it achieves this using the second largest number of bins when summed across all problem instances. In contrast ADJD, which comes 4^{th} when ranked in terms of the number of optimal solutions found, achieves 2^{nd} best position if ranked by the total number of bins required.

Tables 3.3, 7.6 and 3.5 show for problem sets A, B and C respectively the performance of the 4 deterministic heuristics using the number of bins more than optimal as a metric. The tables are subdivided into 11 columns showing for each heuristic the number of problems solved using 0 through to 9 extra bins. Any problems that require ≥ 10 extra bins than the known optimal are grouped into the final column.

Table 3.4: Extra bins (δ) required by 4 deterministic heuristics on the 3968 problem instances from Problem Set B when compared to the known optimal values

	Number of Problems Solved Requiring δ extra bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
FFD	491	2364	442	208	196	51	22	34	68	19	73
DJD	920	1552	468	248	191	100	92	66	57	34	240
DJT	1158	1936	414	141	85	76	52	35	9	2	60
ADJD	**1279**	2398	209	38	33	8	2	1	0	0	0

Table 3.5: Extra bins (δ) required by 4 deterministic heuristics on the 15830 "hard" problem instances from Problem Set C when compared to the known optimal values

	Number of Problems Solved Requiring δ extra bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
FFD	0	8887	2819	1158	1262	365	148	214	405	118	454
DJD	0	7594	2390	1316	1029	581	581	329	282	233	1495
DJT	0	10844	2188	824	577	387	318	199	59	30	404
ADJD	0	14031	1299	228	175	69	18	6	3	1	0

A number of observations can be made from the data in these 3 tables.

- FFD, which is often used as a benchmark heuristic in the literature, performs badly on many problem instances requiring greater than 9 bins extra than the optimal number in 50 out of the 1370 instances in Problem Set A.
- The quality of DJT when evaluated on a larger more diverse set of problem instances than in the publication it was introduced starts to deteriorate.
- Only ADJD, introduced here, solves each of the 21,168 instances included in the three problem sets using less than 10 extra bins.

Table 3.6 summarises the results shown in Tables 3.3, 7.6 and 3.5 showing the total number of bins required by each heuristic for each problem set.

Table 3.6: Total bins required for each benchmark heuristic on each benchmark problem set

	Total Bins Used (% extra) on Problem Set		
Heuristic	A: Optimal = 120433	B : Optimal = 320445	C : Optimal = 1362542
FFD	122575 (1.78 %)	327563 (2.22 %)	1401192 (2.84 %)
DJD	122842 (2.00 %)	330447 (3.12 %)	1419374 (4.17 %)
DJDT	**121314 (0.73 %)**	325743 (1.65 %)	1393531 (2.27 %)
ADJD	121785 (1.12 %)	**323566 (0.97 %)**	**1381083 (1.36 %)**

Table 3.7 shows how the heuristics perform on each problem set if evaluated using Falkenauer's fitness function. The table shows the number of instances for which each heuristic was best using this metric. Note that for many problem instances the best solution is produced by more than one heuristic. The metric used to evaluate the solutions produced by a heuristic determines the ranking of a heuristic. For example on Problem Set A, if evaluated using the number of optimal solutions found as a metric then FFD ranks second best amongst the 4 heuristics solving 788 of the 1370 problem instances. If ranked using the total number of bins required to solve all problem instances then

FFD ranks 3^{rd} out of 4. Using Falkenauer's fitness metric then FFD ranks last. This can be explained by the fact that many of the benchmark problem instances can be described as easy for FFD (and the other heuristics) which finds solutions to many of the problems using the optimal number of bins.However the solutions produced are not best if ranked by Falkenauer's equation as the free capacity is spread between more bins. Whilst the objective of the BPP is ultimately to minimise the number of bins required, solutions where as many bins are packed to capacity have more potential to allow further items to be packed. The heuristics used throughout this study were originally designed as off-line heuristics where no further items are introduced once the packing process begins. For many real world applications the case exists where a combination of off-line and on-line approaches may be most appropriate. An example would be a packing problem where orders were received in batches and bins are despatched as they are filled. In this scenario limiting the free capacity to the last bin to be packed maximises the potential to fully utilise that bin when the next batch of items arrives. In subsequent chapters final results are typically presented using the number of bins greater than the optimal number as a metric. However during the run of the hyper-heuristic developed in the following Chapter and whenever a higher level of precision was required, Falkenauer's equation is employed to ascertain solution quality.

Table 3.7: Ratio of instances in each problem set for which each benchmark heuristic provides the best solution if measured using Falkenauer's fitness function

	Number of Instances (percentage) that the Heuristic was Best or Equal Best On		
Heuristic	Problem Set A (1370)	Problem Set B (3968)	Problem Set C (15830)
FFD	408 (29.78 %)	523 (13.18%)	1235 (7.80 %)
DJD	687 (50.15 %)	1763 (44.43 %)	5691 (35.95 %)
DJDT	**791 (57.74 %)**	2224 (56.05 %)	7006 (44.26 %)
ADJD	764 (55.77 %)	**3348 (84.38 %)**	**13375 (84.49 %)**

3.5.2 Problem Characteristics Vs Difficulty

There are many studies to be found in the literature that talk of problem difficulty in relation to specific problem characteristics and others that measure a problem instances' difficulty based on the success of a particular heuristic [40, 44, 57, 104, 105]. This section investigates these claims in order to ascertain if a direct relationship between problem characteristics and difficulty can be made for a particular algorithm which would facilitate development of a selective hyper-heuristic that maps the characteristics of a problem instance to the best heuristic for solving it. It should be noted that "difficulty" refers to the ability to find the optimal solution from the perspective of a particular algorithm and does not infer that all heuristics will find a particular problem instance equally taxing.

A number of parameters were derived from the literature and investigated.

- Average item size in relation to the bin capacity.
- Total free space summed across all bins as a ratio of the bin capacity.
- The number of distinct integer values that make up the problem instance's item lengths.
- The ratio of items in the ranges small, medium, large and huge as defined in [101].

3.5.3 Average Items per Bin Vs Difficulty

Both Falkenauer [40] and Schwerin and Wäscher [105] comment on the difficulty of triplet problems where the average weight $\varpi = {}^{C}/_{3}$. Schwerin and Wäscher [105] found that problems where $\varpi \rightarrow {}^{C}/_{i} \ : \ \forall \, i \ \in \mathbb{Z}$ are also difficult for FFD and MTP. Another characteristic observed is described as the "variability factor". Schwerin and Wäscher [105] show empirically that the difficulty of a problem instance for FFD decreases as the range of values from which weights are selected increases. Thus problems generated in the range $\omega_j \in [0.25C, 0.4C]$ giving $\varpi = 0.325$ prove more difficult for FFD than those with the same average weight but generated over a wider range

such as $\varpi \in [0.1C, 0.75C]$. For FFD problems where the weights deviate minimally from an average of one third of the capacity of one bin are the most problematic.

Table 3.8 shows the number of optimal solutions found when FFD is used to solve the problem instances from ds2, which are the most varied in terms of item weight. The results shown reinforce the claims made in Schwerin and Wäscher [105] that problem instances with item sizes that deviate little from the average weight are the hardest for FFD with those with an average weight of $\frac{C}{3}$ proving the hardest.

Schwerin and Wäscher [105] go as far as to classify problem sets based on the number of problem instances solved optimally using FFD. They define problem sets as shown below.

- $Easy\ p \geq 80\%$
- $Hard\ 80\% > p \leq 20\%$
- $Extreme\ p < 20\%$

Out of the 48 parameter combinations used to create ds2, 15 classes can be considered easy, 16 hard and 17 extreme. The results obtained show a correlation between the number of items in a problem and the ability of FFD to find an optimal solution. A slight anomaly is observed between results obtained on problem instances generated using a wider weight distribution (deviation of 90% from the average) with 500 items which proved less difficult to solve than problem instances comprising of only 200 items. If plotted over a wider range of problems the figures become less clear. Figure 3.4 plots, for all 1370 problem instances from Problem Set A, the mean number of items in a bin in each solution obtained by solving each instance using FFD, DJD, DJT and ADJD.

The four heuristics don't all behave the same and have different capabilities with DJD performing badly as the average item size decreases and FFD performing the worst on the "hard triplet" problems. All four plots show peaks of varying magnitude at exactly 3, 5 7 and 9 reinforcing the observations made by Schwerin and Wäscher [105] that those problem instances are most problematic[1].

[1]Note that if the average size is an exact integer value then the free space summed across all bins must equal 0 which is examined in the following section.

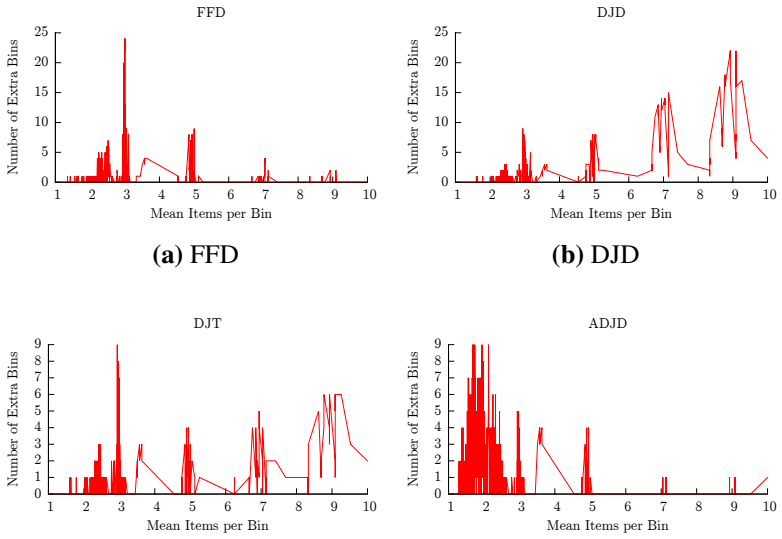
Table 3.8: Ratio of ds2 Solved by FFD which is used as a measure of problem difficulty in many publications

		Deviation of Weight From Average		
Number of Items	Average Weight	20%	50%	90%
50	$C/3$	0	0	70
	$C/5$	40	70	100
	$C/7$	70	90	100
	$C/9$	100	80	100
100	$C/3$	0	0	60
	$C/5$	0	60	90
	$C/7$	70	90	100
	$C/9$	70	70	90
200	$C/3$	0	0	50
	$C/5$	0	10	70
	$C/7$	0	60	90
	$C/9$	20	50	100
500	$C/3$	0	0	60
	$C/5$	0	0	80
	$C/7$	0	0	90
	$C/9$	0	60	100

3.5.4 Number of Items Vs Difficulty

As is the case with all grouping problems an increase in the number of items in a problem instance causes the number of possible permutations to rise exponentially. For all the benchmark problem instances studied, with the exception of $ds3$, the item weights and bin capacities are relatively small leading to an increase in the number of duplicate item weights as the number of items increases. As items of the same weight may be interchanged, having multiple items of the same weight reduces the number of discrete combinations that need to be considered. Figure 3.5 plots, for all 1370 problem instances in Problem Set A, the number of items in each problem instance against the number of extra bins required by each of the heuristics FFD, DJD, DJT and ADJD.

All plots show a peak at $\varpi = 500$ which is more pronounced for Figures 3.5a and 3.5b corresponding to FFD and DJD respectively. Examining the worst cases shows that all 20 problem instances from $FalU$ with $n = 501$ require more than 15 extra



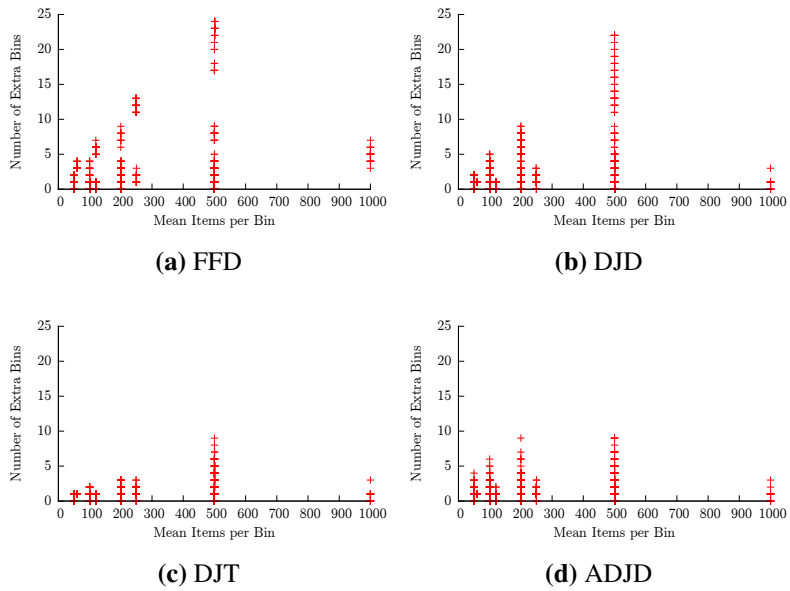
(a) FFD **(b)** DJD

(c) DJT **(d)** ADJD

Figure 3.4: Heuristic Performance Compared to Average Item Size. DJD and DJT both show worsening performance as the average number of items increases. Conversely, the opposite is true for FFD and ADJD which perform well on problem instances with greater numbers of items in each bin.

bins when solved with FFD but can be solved using less than a third of this number by the other heuristics. Similarly FFD uses more than 15 bins extra for all problems from $ds2$ with 500 items with $\varpi = {}^{c}/_{3}$ and $\delta = 20\%$ backing up the assertion made previously that the triplet problems are the most troublesome for FFD. There is an obvious increase in computational power needed as the number of items in a problem instance rises. The graphs depicted in Figure 3.5 do show a decrease in solution quality as the number of items increases but this is also largely due to factors associated with other characteristics and is not a constant correlation across all problem instances. To highlight this Figure 3.6a plots the quality of the solutions attained by FFD on only those problem instances from $ds2$ that were generated using $\varpi = {}^{C}/_{7}$ and $\delta = 50\%$. Figure 3.6b plots the quality of the solutions attained by FFD on only those problem instances from $ds2$ that were generated using $\varpi = {}^{C}/_{9}$ and $\delta = 90\%$ Both subsets comprise of 40 problem instance that vary only in $n \in \{50, 100, 200, 500\}$

It is clear from both plots that the number of items in the problem cannot be used in



(a) FFD **(b)** DJD

(c) DJT **(d)** ADJD

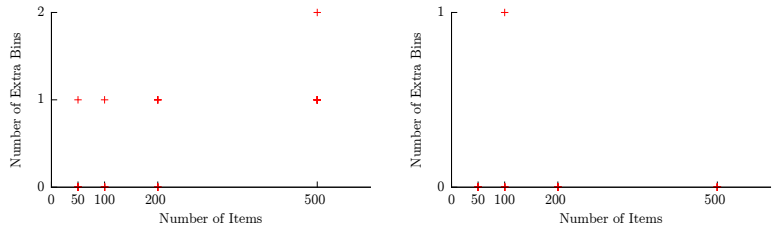
Figure 3.5: Heuristic Performance Compared to Number of Items. Each plot shows the average number of bins more than optimal obtained by each heuristic Vs the mean number of items per bin in the optimal solution for all 1370 problem instances from Problem Set A

isolation to determine the success a heuristic will have and that other parameters must also be taken into consideration.

3.5.5 Free Space Vs Difficulty

The observation made in [105] that instances where the average weight, $\varpi_j \rightarrow \frac{c}{3}$ are the most difficult with those where $\varpi_j \rightarrow \frac{c}{4}, \frac{c}{5}, \frac{c}{6} \ldots$ proving difficult also can be expressed using the amount of free space summed across all bins. If a solution exists at the lower bound given in Equation 2.1 then the total free space $\varpi_{free} \rightarrow 0 \; as \; \varpi_j \rightarrow \frac{c}{i} : i \in \mathbb{N} : i \geq 3$

Figure 3.7 shows the amount of free space summed across all bins as a ratio of the bin capacity plotted against the number of extra bins required for each instance. There appears no apparent correlation. The exception is a distinct line at the origin in Figure 3.7a for FFD which is as a result of Falkenaur's triplet problems which have



(a) FFD Vs Item Size $\varpi = {}^{C}/_{7}$: $\omega = 50\%$

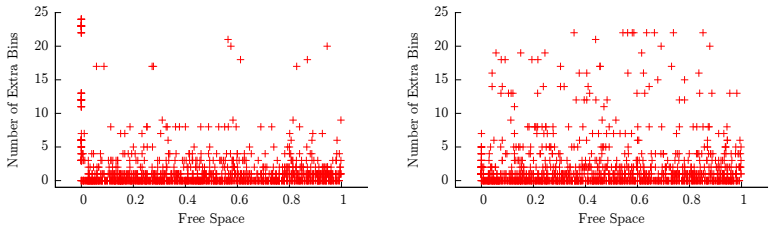
(b) FFD Vs Item Size $\varpi = {}^{C}/_{9}$: $\omega = 90\%$

Figure 3.6: FFD Performance Compared to Number of Items in ds2. Figure 3.6a plots the quality of the solutions attained by FFD on only those problem instances from $ds2$ that were generated using $\varpi = {}^{C}/_{7}$ and $\delta = 50\%$. Figure 3.6b plots the quality of the solutions attained by FFD on only those problem instances from $ds2$ that were generated using $\varpi = {}^{C}/_{9}$ and $\delta = 90\%$ Both subsets comprise of 40 problem instance that vary only in $n \in \{50, 100, 200, 500\}$

optimal solutions where no free space exists in any bin. It appears that where there is no free space in an optimal solution the difficulty of finding such a solution increases. However if this characteristic is relaxed even slightly the ability to find an optimal solution has little correlation.

As with the other characteristics investigated other factors mask any increase in difficulty associated with the free space when plotted on the complete set of problem instances. The newly generated Problem Set C, described in Section 3.3.7, consists of 15830 problem instances of which 78% have optimal solutions with no free capacity in any bin. None of the four heuristics investigated can find optimal solutions to any of these problems (the problem instances were selected for this feature). However the ability to find solutions using only one bin greater than optimal appears unaffected. As an example ADJD finds solutions to 89% of the problems in Problem Set C using only 1 bin more than the optimal number. In contrast, on Problem Set A, even though ADJD solves 50% of all 1370 problem instances using the known optimal number of bins it only manages to solve a further 27% using at most 1 bin greater than the optimal number or cumulatively 77% of all problems in the set using at most 1 extra bin. This highlights an underlying feature of the search space which appears to have

many plateaus that offer only marginally sub optimal solution quality. Figure 3.7 is replicated in 3.8 for Problem Set C and heuristic FFD.



(a) FFD **(b)** DJD

(c) DJT **(d)** ADJD

Figure 3.7: Benchmark heuristics performance compared to the total free space in the optimal solution. DJT and ADJD perform more consistently across the complete set of problems. Only the plot for FFD highlights the difficulty of obtaining an optimal solution as the free space is reduced to 0.

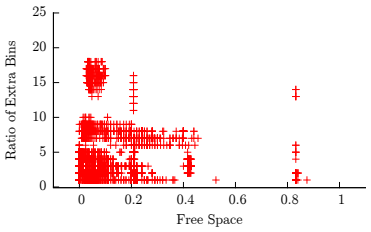
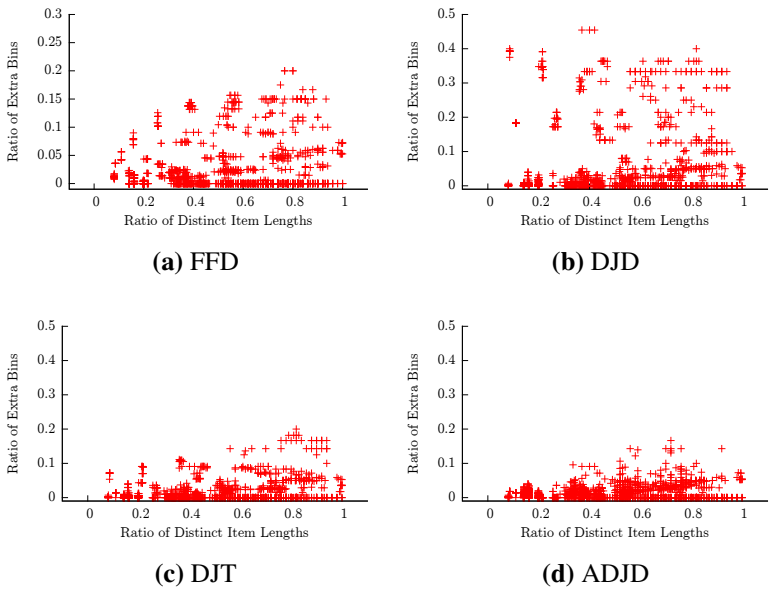


Figure 3.8: FFD Performance compared to the total free space in the optimal solution for problem set C

3.5.6 Distinct Items Vs Difficulty

In [104] the number of distinct integer values occurring in a problem instance was determined to be a contributory factor to the difficulty of finding an optimal solution. It is clearly the case that for problem instances where there are multiple items of equal size that these items are interchangeable and therefore the number of distinguishable solutions is reduced.

The majority (800) of the problem instances in Problem Set A have a bin capacity of 150 or less with item weights drawn from a uniform distribution covering only a fraction of the total capacity. This results in problem instances where many of the item weights are duplicated which consequently, if the theory is correct, makes the problem instances less difficult. The 10 "hard" problem instances in $ds3$ introduced inScholl et al. [104] and described in Section 3.3.5 are generated with 200 items each with weights sampled from the range $\omega \in [20000, 35000]$ for a fixed capacity $C = 100000$. The authors assert that this feature makes these problem instances more difficult than a similar problem instance with fewer distinct item weights. None of the human designed heuristics described here are able to find an optimal solution to any of these 10 problem instances requiring between 3.6% (DJD and DJT) and 6% (FFD) more bins than the optimal when summed across all 10 problems. This is less of a spread than is exhibited on other problem instances with similar item weight to bin capacity ratios. These instances have been solved successfully using different techniques including hyper-heuristics [20]. It should be noted that a second feature of these problems is that the items are drawn from a uniform distribution which varies by only $0.075C$ either side of the average weight of $0.275C$. As noted earlier problem instances with little variance in item weight with an average weight of $\frac{C}{3}$ are the most difficult. Figure 3.9 plots the ratio of extra bins required by each heuristic against the ratio of distinct items. There is a general increase in difficulty for all but DJD as the ratio of distinct items increases with the most pronounced effect seen for FFD.



(a) FFD **(b)** DJD

(c) DJT **(d)** ADJD

Figure 3.9: Benchmark heuristics performance compared to the number of distinct items. Each plot shows how a different heuristic performs in relation to the ratio of distinct integer values in a problem across all problems in problem set A.

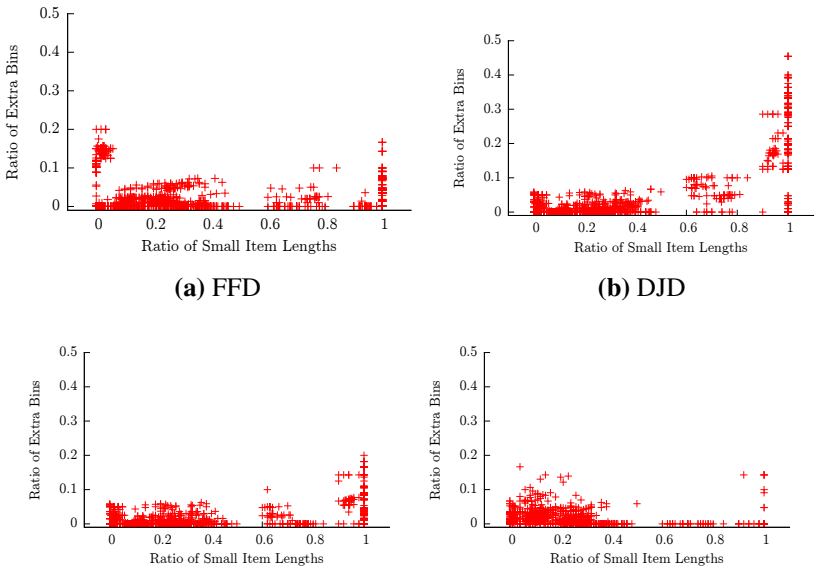
3.5.7 Item Weight Ranges Vs Difficulty

In Section 3.5.3 the performance of each heuristic was examined in relation to the average item weight. Of all of the characteristics examined the average weight could be argued as being the best single indicator for predicting the success of a heuristic. Although generalisations can be derived about the performance of a particular heuristic, using this coarse grained characteristic fails to encapsulate more detailed information such as relationships between items of different sizes. In Section 3.5.5 it was identified that problems with average item weights of $\frac{1}{3}C$ prove the most troublesome for heuristics such as FFD. This difficulty increases for problem instances with item sizes that vary the least from the average. Using only the average weight fails to encapsulate this information. In order to increase the level of information the performance of each heuristic is described here in relation to the ratio of items in a problem with item weights in each of 4 ranges, taken from [101] and described briefly in Section 3.3.7 where the ranges were used as parameters to generate new problem instances. Figures

3.10 through 3.13 plot the ratio of extra bins required for each heuristic in relation to the ratio of items falling into each of these 4 ranges described below.

- Small items of size $\omega \leq \frac{C}{4}$
- Medium items of size $\frac{C}{4} < \omega \leq \frac{C}{3}$
- Large items of size $\frac{C}{3} < \omega \leq \frac{C}{2}$
- Huge items of size $\omega > \frac{C}{2}$

The authors of [101] identified these ranges as being natural choices to be used to predict problem difficulty and heuristic solution quality due to the fact that only 1 huge, 2 large or 3 medium items can be placed into the same bin together. These characteristics have been exploited by authors of a number of hyper-heuristic approaches that have been applied to a range of packing problems of different dimensionality [102, 124, 125, 128].



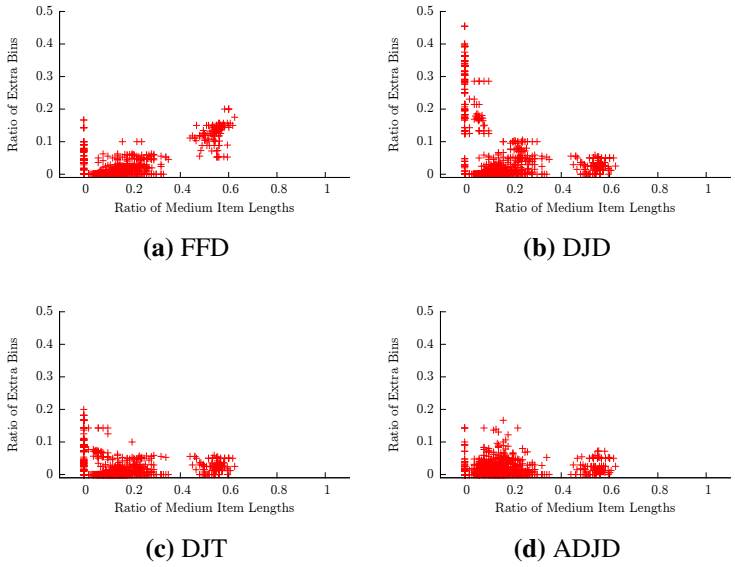
(a) FFD **(b)** DJD

(c) DJT **(d)** ADJD

Figure 3.10: Benchmark heuristics performance compared to the number of *small* items in a problem for all problems in Problem Set A. Small items are defined as those with $\omega \leq \frac{C}{4}$. Results are plotted for all 1370 problem instances in Problem Set A

Figure 3.10 highlights the poor performance of DJD as the number of small items in a problem increases which was noted previously in Section 3.5.3. This is easily explained by the heuristics description. Once DJD has filled a bin to $\frac{1}{3}C$ it searches for at most three items to fill the remaining space. As item sizes get smaller the ability to find three items to fill the remaining $\frac{2}{3}C$ space decreases.

The remaining plots show this effect mirrored for DJD at the opposite end of the scale where few large items are included. In contrast ADJD proves to be the best performing algorithm on this class of problem.



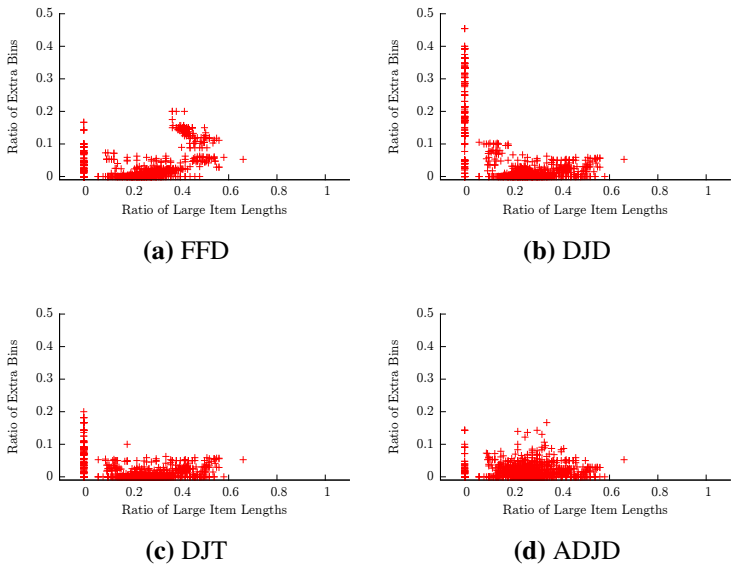
(a) FFD **(b)** DJD

(c) DJT **(d)** ADJD

Figure 3.11: Benchmark heuristics performance compared to the number of *medium* items. Medium sized items are defined as those with $\frac{C}{4} < \omega \leq \frac{C}{3}$. Results are plotted for all 1370 problem instances in Problem Set A

Figure 3.11 shows little correlation between the number of medium sized items and problem complexity. There appears a slight improvement in the solution quality produced by all heuristics except FFD as the ratio of medium items increases.

Again, in Figure 3.12, there is little apparent correlation when looking at the ratio of large items in isolation. The peaks shown at 0 can be explained by Falkenauer's triplet problems which prove consistently difficult for all of the heuristics investigated.

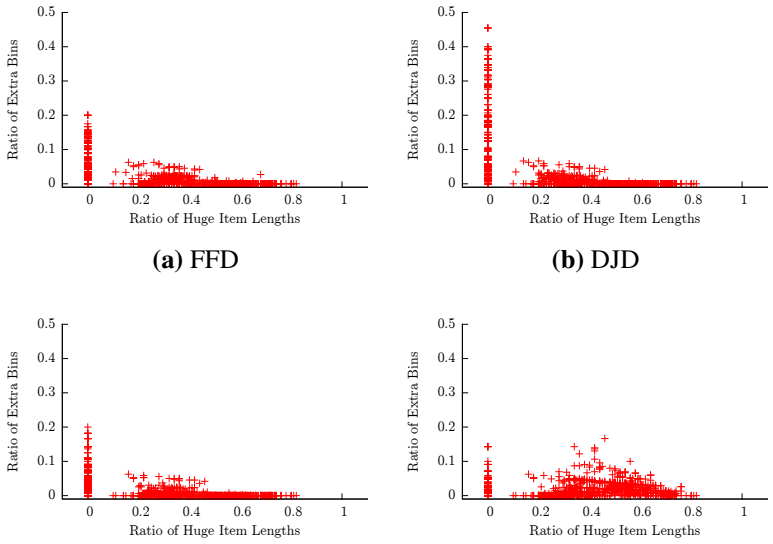


(a) FFD **(b)** DJD

(c) DJT **(d)** ADJD

Figure 3.12: Benchmark heuristics performance compared to the number of *large* items. Large sized items are defined as those with $\frac{C}{3} < \omega \leq \frac{C}{2}$. Results are plotted for all 1370 problem instances in Problem Set A

Examining the ratio of huge items in isolation shows little correlation as would be expected as each huge item has to be placed into a seperate bin which can only be filled if there are sufficient numbers of smaller items to fill the remaining capacity.



(a) FFD **(b)** DJD

(c) DJT **(d)** ADJD

Figure 3.13: Benchmark heuristics performance compared to the number of *huge* items. Huge sized items are defined as those with $\frac{C}{2} < \omega \leq C$. Results are plotted for all 1370 problem instances in Problem Set A

3.6 Summary

It is clear that relationships do exist between certain problem characteristics and the quality of the solutions produced by individual heuristics. It is also apparent that these relationships differ depending on the heuristic that is used. This knowledge asserts the hypothesis that no heuristic can perform better than all others on all possible problems from a particular domain and reinforces the potential that hyper-heuristics offer in exploiting the strengths that individual heuristics exhibit on different niche areas of the problem landscape. Although relationships have been shown to exist between a heuristics performance and the characteristics of the problem instances that it works best on these are not simple and there is no proof given that the characteristics chosen are the most relevant. Correlations appear to be dependant on complex combinations of different characteristics that vary from the perspective of each heuristic and any assertions made here are at best only general in their nature with many problem instances causing behaviour in heuristics that varies from the norm. It is difficult to identify any

definitive correlations that exist between any single characteristic and the performance of an individual heuristic other than broad generalisations and therefore, the ability to predict a heuristic's performance based upon a single characteristic of a particular problem instance is likely to by successful only in the most general of cases. However, there are deviations in different heuristics' performance's on different parts of the problem landscape which could be exploited by a selective hyper-heuristic. The single characteristic that appears the best predictor of solution quality appears to be the average item weight. Subdividing this characteristic into four and viewing the ratio of items in each range in isolation does not improve upon any observed correlation and no assertions can be made by looking at these characteristics in isolation.

The following chapter starts by exploring the utility to be gained by choosing between a range of different heuristics for a broad range of problem instances of widely differing characteristics. A novel selective hyper-heuristic is then introduced which attempts to derive relationships between different combinations of characteristics and the solution quality obtained from the perspective of 4 benchmark heuristics.

Chapter 4

Selective Hyper-heuristics

Hyper-heuristics have been categorised into two main classes [19]. Selective hyper-heuristics choose the best heuristic (or combination of heuristics) for a problem instance from a pool of predefined heuristics. Generative hyper-heuristics automate the heuristic design process, typically by evolving new heuristics from constituent heuristic components using Genetic Programming. The previous chapter explored the performance of a set of simple deterministic heuristics with respect to a number of problem characteristics and showed that the quality of solutions attained by individual heuristics varies with changes in certain characteristics of the problems they are being used to solve. This chapter explores the utility to be gained by selecting the best heuristic from this set of simple heuristics when attempting to solve a large set of diverse problem instances and introduces a selective hyper-heuristic that attempts to exploit this potential by predicting which heuristic will produce the best solution. As noted in the previous chapter, the task of finding a mapping between problem characteristics and the utility of a particular heuristic has been investigated by others [117, 118, 119] since the algorithm selection problem was introduced in the seminal work of Rice [99]. The work presented here differs in that the set of attributes used to describe a problem instance is not fixed. In an attempt to improve the prediction accuracy of an off the shelf classification algorithm, an EA is used to evolve the set of characteristics (predictor attributes) passed to the classifier rather than relying on features that are deemed important by the human-designer. Rather than using a single problem characteristic in

isolation the classification algorithm is supplied with a vector describing a number of problem characteristics.The characteristics supplied to the classification algorithm are not predetermined or fixed in size but are an emergent property of the hyper-heuristic which employs a messy evolutionary algorithm in an attempt to derive the characteristics that most affect each heuristic.

4.1 Contribution

This chapter describes a novel selective hyper-heuristic that uses a $k-$nearest neighbour classification algorithm to predict which from a set of deterministic constructive heuristics will perform best on each of a large set of Bin Packing Problem (BPP) instances. The hyper-heuristic presented uses half of a large set of 1370 problem instances to train a classification algorithm which is then used to predict the most suitable heuristic for each of the other unseen test problem instances. Rather than using pre-determined characteristics of the problem, an Evolutionary Algorithm (EA) is incorporated which is used to evolve a set of predictor attributes that best map to a heuristics performance on the set of problem instances and improve the prediction accuracy obtained using the classification algorithm. The EA evolves divisions of variable quantity and dimension that represent ranges of item length expressed as a ratio of a bin's capacity. The ratios of items with lengths specified by each range are used as predictor attributes to train a k-nearest neighbour algorithm with the accuracy attained during training used as the quality metric for the EA. The evolved classifier is shown to achieve results significantly better than are obtained by any of the constituent heuristics when used in isolation. This chapter is inspired by research initiated by Ross, Schulenburg, Marín-Blázquez, and Hart in [101] and [102] and was published in [115].

4.2 Background and Motivation

Selective hyper-heuristics aim to exploit the strengths of individual heuristics by selecting the best heuristic or combination of heuristic components for the problem requiring

to be solved. The literature surrounding selective hyper-heuristics, reviewed previously in Section 2.3.2, describes methods that iteratively apply perturbative heuristics in order to improve an already valid solution and methods that use combinations of constructive heuristics to incrementally build a solution. The approach taken here is simply to select the single most appropriate constructive heuristic for each of a large set of problem instances and use to to generate a complete solution. The utility of the approach is in predicting which heuristic to should be used for a given problem instance.

The previous chapter investigated how a heuristics performance varies with respect to a selection of predetermined characteristics of the problem instance presented to it. The motivation here is to determine the extent to which the variance shown by different heuristics can be exploited and whether the performance of individual heuristics can be predicted based on characteristics of the problem instance to be solved. For a selective hyper-heuristic approach to be successful the premise that different heuristics perform differently on different problem instances must be exploited. 1370 problem instances are used to evaluate the approach presented here and are described in the previous Chapter in Sections 3.3.3 and 3.3.5.

Four heuristics (FFD, DJD, DJT and ADJD) described in Section 3.4 were included in the system. All are deterministic off-line heuristics that require that a problem instance's items are presorted in decreasing weight order.

The remainder of this chapter attempts to address two fundamental assertions that underpin selective hyper-heuristic research.

- To what extent does selecting the best from a pool of simple deterministic heuristics improve the solution quality obtained when applied to a large diverse set of problem instances compared to the abilities of the individual heuristics?

- Can the best heuristic be predicted for each instance based on characteristics derived from each problem instance?

The following section addresses the first and simpler of these two questions, analysing the combined performance of the 4 deterministic heuristics used when applied greedily to all 1370 benchmark BPP instances. As the hyper-heuristic presented later in the

chapter applies a single heuristic to each problem instance without modification, the results presented in the next section represent the optimal results that could be obtained if the hyper-heuristic were to select the best heuristic for each problem instance.

4.3 Potential of Combining Heuristics

Hyper-heuristics search a landscape defined by the performance of the heuristics they encompass on a given set of problem instances. For a selective hyper-heuristic to be effective the set of heuristics used must be able to collectively outperform the individual heuristics; the combination of heuristics must cover the problem space better than the component heuristic parts. Section 3.5 shows that the heuristics identified in Section 3.4 perform differently on the set of problem instances identified in Section 3.3 and that a heuristics performance varies relative to certain problem characteristics. This section expands on the previous chapter by investigating how the *s*et of heuristics perform (when combined using a greedy selection strategy) compared to the performance of the individual heuristics when applied to a broad range of problem instances.

In Chapter 3 Table 3.2 the performance of four deterministic heuristics on 5 data sets totalling 1370 problem instances was summarised for each heuristic and each data set. Table 4.1 replicates this information but appends the combined results achieved by the set of four heuristics; the results obtained if the best heuristic is chosen for each instance. It is clear that whilst individual heuristics dominate others on individual data sets, when measured against the complete set of problems their relative performances start to level out. The performance of these four heuristics, when applied greedily to a large diverse set of problem instances, outperforms the ability of any of the heuristics when applied in isolation. This knowledge shows that even a simple selective hyper-heuristic using a minimal set of heuristics can prove fruitful.

The performance of these heuristics can be dissected further. Each of the data sets summarised in Table 4.1, with the exception of $ds3$, is comprised of problem instances generated using a variety of parameter settings. These parameter settings were described previously in Table 3.1, Section 3.3.6. For $ds1$ there were 36 different parameter combinations used with 20 problem instances generated from each combi-

Table 4.1: The table shows the results obtained by each heuristic on different data sets using two metrics; The percentage of problems solved using the optimum number of bins and the percentage of extra bins required over the optimal number. The headings in the first row depict the data sets as described in Table 3.1 with the column headed All representing the complete set of 1370 instances.

	ds1 720 Problems		ds2 480 Problems		ds3 10 Problems		falk U 80 Problems		falk T 80 Problems		All 1370 Problems	
	solved	bins	solved	bins	solved	bins	solved	bins	solved	bins	solved	bins
FFD	75.83	0.36	49.17	3.69	0	6.05	7.5	1.3	0	14.21	57.52	1.78
DJD	79.03	0.28	21.04	9.77	0	3.56	57.5	0.27	0	2.45	52.26	2.00
DJT	83.75	0.17	44.58	2.66	0	3.56	57.5	0.27	0	2.45	62.99	0.73
ADJD	35.83	1.32	80.21	0.66	0	5.87	53.75	0.33	0	1.68	50.07	1.12
Combined Greedy	90.56	0.10	81.25	0.64	0	3.56	60.00	0.24	0	1.68	79.56	0.30

nation resulting in 720 problem instances. The problem instances in $ds2$ were created in groups of 10 using 36 different parameter combinations. $ds3$ consists of 10 problem instances generated using a single set of parameters. Falkenauer used 4 parameter combinations for each of his 2 data sets with 20 problem instances generated for each combination.

Each of the graphics depicted in Figure 4.1 compares the performance of 2 different heuristics on the full set of 1370 problem instances used in this study. Each graphic contrasts the performance of a pair of heuristics with each row indicating a set of either 10 or 20 problem instances that were generated with the same parameters as described in Table 3.1. Each cell depicts a different single problem instance. The colour indicates which of the two heuristics under comparison performs best on each instance if evaluated using Equation 3.1. Uncoloured cells represent those problem instances for which both heuristics generate identical quality solutions when evaluated using Equation 3.1.

It is clear, in many cases, that a heuristic which performs best on a certain problem instance often produces the best solution when applied to problem instances generated using the same parameter settings. In Figure 4.1 this is highlighted by the fact that many contiguous cells, or problem instances, are solved best by the same heuris-

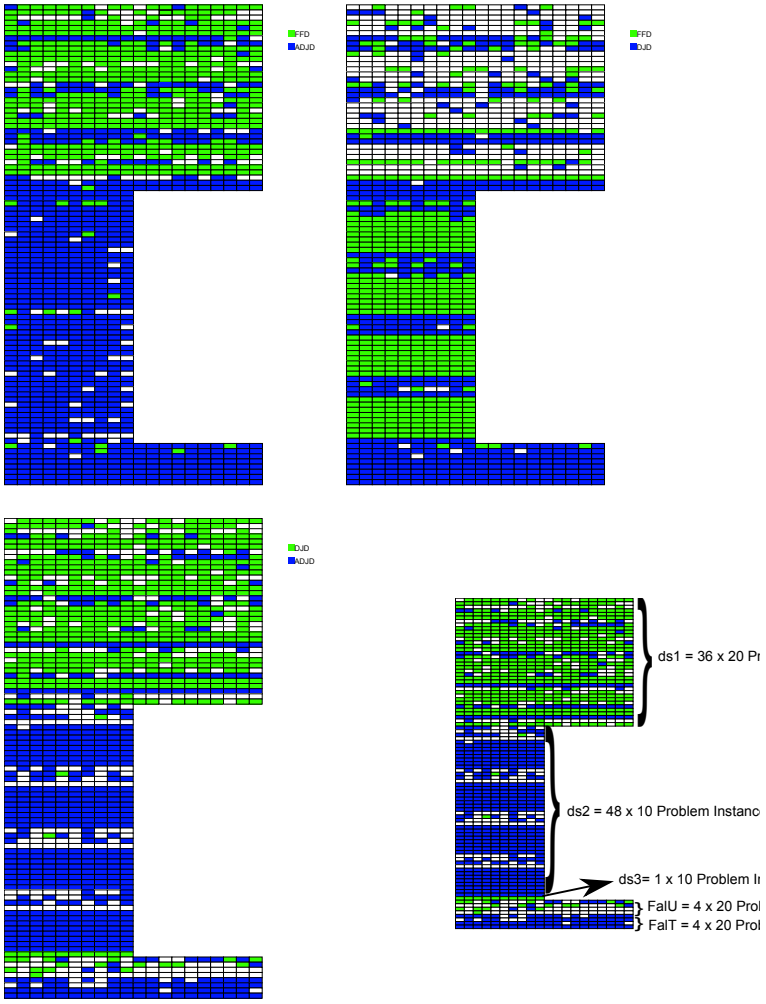


Figure 4.1: A comparison of the relative solution quality attained by different pairs of benchmark heuristics. The three larger diagrams compare FFD Vs ADJD, FFD Vs DJD and DJD and ADJD. The last sub figure illustrates how the 5 data sets described in Table 3.1 that make up the problem instances from Problem Set A are arranged in the diagrams. All problem instances in each row are generated using the same item weight distributions. Coloured cells highlight those problem instances that are solved best by a single heuristic when measured using Falkenauer's fitness function whereas white cells correspond to problem instances where the best solution is attained using more than one heuristic.

tic. Choosing the best heuristic for each problem instance clearly increases the overall quality of the solutions obtained when contrasted with the ability of any single heuris-

tic. Observing the patterns exhibited in the visualisations there appears to be a relationship between the quality of a solution obtained by a heuristic and the characteristics, or parameters, used to generate the problem instance.

This distinction between heuristics performance becomes less apparent when more than two heuristics are compared. Figure 4.2 shows the relative performance of all 4 heuristics investigated. Many of the problem instances are now solved equally by more than one heuristic, especially those included in $ds1$. The new heuristic introduced in the previous Chapter, ADJD, dominates on the problem instances from $ds2$. Although the distinction is less clear, the combined performance of the four heuristics is substantially greater than any constituent heuristic as is shown in Table 4.1. The best individual heuristic, DJT, is able to solve optimally 62.99% of the 1370 problem instances. The collection of heuristics finds optimal solutions to 79.56% which rises to 85.16% if the deliberately designed "hard" instances from $ds3$ and $FalU$ are omitted. None of the heuristics investigated here is able to find an optimal solution to any of these two subsets of problem instances. Similarly, if contrasted using the percentage of extra bins greater than the optimal number of 120433, the best single heuristic is again DJT which requires an extra 0.73% (881) bins. If the best heuristic is chosen greedily for each problem instance this number falls to 0.3% greater than the optimal number of bins (361).

The relative performances of different heuristics on different problem instances indicates the potential benefit of selecting between different heuristics when presented with problem instances of different characteristics. The ability of a selective hyper-heuristic to efficiently take advantage of the individual strengths of heuristics is however reliant on discovering a relationship that allows a mapping to be made between heuristic and problem instance.

As covered in Section 3.2, in order to get a better indication of a heuristic's performance than can be deduced from either the number of optimal solutions found or the number of extra bins required, Falkenauer's fitness function, given in Equation 3.1 is used with k set to 2 in order to reward solutions where any free capacity is restricted to as few bins as possible. This allows for a distinction to be made between different solutions to the same problem instance that use an equal number of bins and increases

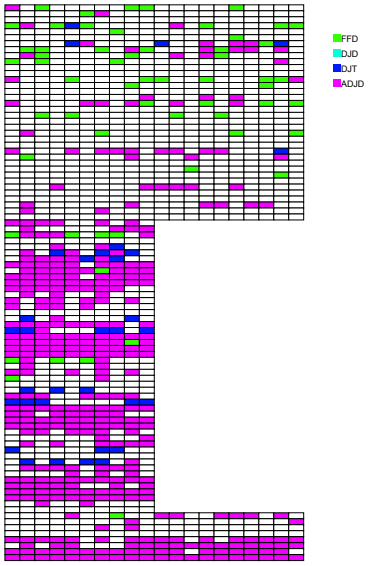


Figure 4.2: A comparison of the relative solution quality of 4 benchmark heuristics on the 1370 problem instances in Problem Set A. Coloured cells highlight those problem instances that are solved best by a single heuristic when measured using Falkenauer's fitness function whereas white cells correspond to problem instances where the best solution is attained using more than one heuristic.

the precision with which a solution's quality can be measured.

Table 4.2 shows, for each data set, the number of times that each heuristic achieves the best solution based on Falkenauer's metric. The table also shows the number of instances for which each heuristic was the single best heuristic; the solution attained was better than produced by any of the other heuristics. In comparing the data presented in Tables 4.1 and 4.2, it is interesting to note for instance, that whilst FFD rates highly if ranked in terms of the number of optimal solutions found, it achieves this using the second largest number of bins. In contrast ADJD, which comes 4^{th} in terms of the number of optimal solutions found, achieves 2^{nd} best position if ranked by either of the other two metrics.

It is apparent from the data presented in Table 4.2 and visualised in Figure 4.2 that the same solution is achieved by more than one of the heuristics in many cases when applied to the problem instances used. However the combined capability of the heuristics chosen when measured using either of the metrics presented in Table 4.1

Table 4.2: Benchmark heuristics performance on benchmark problems using Falkenauer's fitness function

	ds1 720 Problems		ds2 480 Problems		ds3 10 Problems		falk U 80 Problems		falk T 80 Problems		All 1370 Problems	
	best	distinct	best	distinct	best	distinct	best	distinct	best	distinct	best	distinct
FFD	393	73	14	12	0	0	1	1	0	0	408	86
DJD	513	1	96	0	10	0	60	0	8	0	687	1
DJT	552	14	161	40	10	0	60	0	8	0	791	54
ADJD	202	89	425	303	0	0	60	19	77	72	764	483

indicates the merit of using a combined hyper-heuristic strategy. The remainder of this chapter explores whether a mapping can be automatically determined that allows prediction of the heuristic best suited to solving each instance based on characteristics of that problem instance.

4.4 A Selective Hyper-Heuristic

Based on the analysis conducted in the previous chapter the remainder of this chapter investigates whether a relationship can be found that maps a heuristics performance to the characteristics of the problem instance it is trying to solve. As previously mentioned, other authors have tried to define problem characteristics that can be used to categorise heuristics such as in [101, 102] where the authors used predetermined "natural" characteristics to describe problem instances. These included the ratio of a problem's items with weights within 4 different size ranges expressed as ratios of the maximum bin capacity. While these ranges appear to be good choices they fail to encapsulate any information about combinations of different item sizes that may be useful predictors of a heuristics ability.

The approach presented here differs in two key respects. First it does not use predefined categories to describe an instance's state. A messy evolutionary algorithm is used to evolve a set of ranges that when used in conjunction with a classifier algorithm, map the description of an instance to a suitable simple heuristic. Second in contrast to [101], problem instances are only categorised once and solved using a single heuristic as opposed to being reclassified after each item is packed. The motivation behind

this approach is to determine whether it is possible to find an appropriate method of describing a set of problem instances such that each instance can be mapped to the single heuristic that best solves it. The authors of [101] showed that the task of mapping problem characteristics to a suitable heuristic was non-trivial and were unable to find a relationship using a perceptron. The method utilised here attempts to remove the preconceptions imposed by the human designer by utilising an EA to improve the accuracy of the mapping.

Before describing the EA in more detail Figure 4.3 is presented as a conceptual overview of the system. The system incorporates an EA, a classification algorithm, a set of deterministic constructive heuristics and a set of problem instances. The hyper-heuristic uses the classification algorithm as a heuristic selection strategy to choose which from a set of deterministic constructive heuristics to apply to a problem instance based on knowledge of the problem domain obtained during an off-line training phase. The classification algorithm attempts to match an unseen problem instance to a procedure for solving it based on the problem instance's characteristics. The characteristics used are the ratio of an instance's items with weights within a number of ranges, expressed as ratios of the bin capacity. The ranges used are not fixed in number or dimension but are evolved by the EA during a training phase. The accuracy attained on an unseen subset of the training problem instances during each iteration of the training phase is used as the objective fitness measure for the EA.

The 1370 problem instances described in Sections 3.3.3 and 3.3.5 were split in to equal sized training and test sets with every second problem instance used for testing. As the problem instances were generated from 93 parameter combinations with either 10 or 20 instances generated using each parameter setting the split into training and test sets ensures an even ratio of problem instances from each of the data sets and each subset of problem instances generated from each set of parameters. The parameters used to generate these problem instances are summarised previously in Table 3.1.

The system is described in more detail by Figure 4.4 and by Algorithm 1. It comprises of a database containing the problem instances and corresponding solutions attained by each heuristic along with a classification algorithm and an EA. The process of storing the problem instances and the corresponding solutions obtained by each

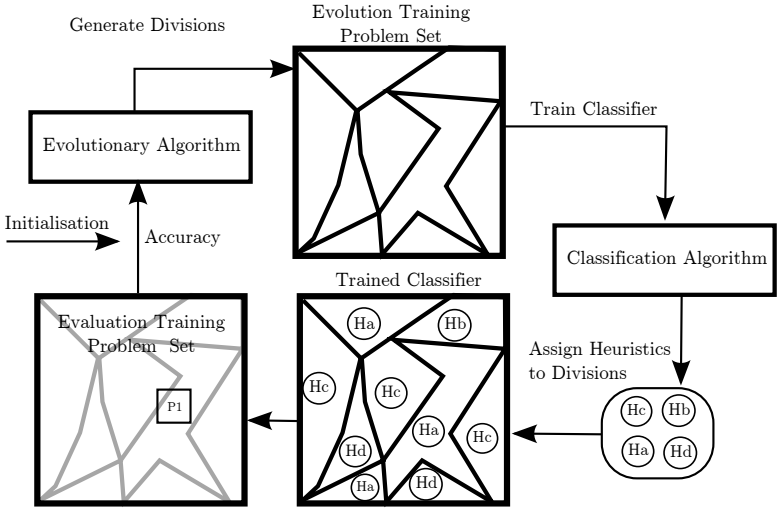


Figure 4.3: During off-line training, the EA generates problem divisions, of varying dimension and number, that the classifier assigns the best known heuristic to. The classifier's accuracy in predicting which is the best heuristic for a set of unseen problem instances is used as feedback to the EA.

heuristic in a database allows for a substantial decrease in the computational resources required while training. As all the problem instances are static and the heuristics deterministic this can be achieved with ease. It is trivial to attain the best set of possible solutions by querying the database, however the purpose is to show that a relationship can be evolved, and that the technique has merit when the problem instances are not known beforehand.

The classifier is used to predict which heuristic will perform best on each of the unseen problem instances whilst the EA attempts to increase classification accuracy during training by evolving the ranges to be used as predictor attributes. Unlike other applications in which classifiers and EAs have been combined to select which predetermined predictor attributes should be used, the approach here uses the EA to evolve combinations of problem characteristics not known *a priori*. For a comprehensive review of EAs combined use with classification algorithms the reader is directed to [49].

During training the 685 problem instances in the training set are further subdivided into *evolution* and *evaluation* sets with every 5^{th} problem placed in the *evaluation* set. The *evolution* set is used to train the classifier using the ranges specified by a

Table 4.3: Sample data passed to the classification algorithm

Huge Items	Large Items	Medium Items	Small Items	Best Heuristic
0.28	0.16	0.12	0.44	DJDK
0.38	0.06	0.22	0.34	FFD
0.26	0.08	0.22	0.44	DJD
0.28	0.06	0.24	0.42	DJD
0.26	0.12	0.18	0.44	FFD
0.26	0.12	0.14	0.48	DJDK
0.28	0.04	0.12	0.56	FFD
0.18	0.04	0.18	0.6	FFD
0.2	0.08	0.18	0.54	FFD
0.24	0.08	0.14	0.54	FFD
0.1	0.1	0.26	0.54	DJD
0.12	0.08	0.22	0.58	DJD

chromosome as the predictor attributes and the best heuristic for each problem instance used as the class we wish to predict. Once built the classifier is used to predict the best heuristic for each instance from the *evaluation* set and the ratio that are correctly predicted are used as the objective fitness value for that chromosome.

A sample of the data passed to the classifier is shown in Table 4.3 for the benchmark chromosome represented by Figure 4.5.

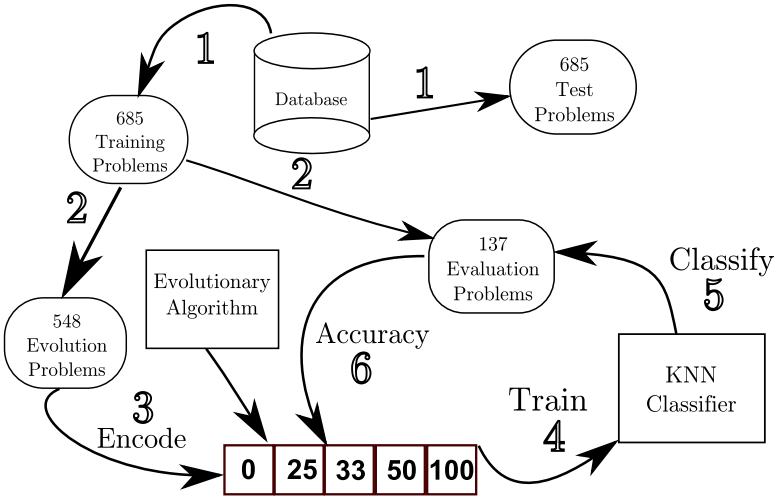


Figure 4.4: The system elements and algorithm steps explained by Algorithm 1

All software, with the exception of the classification algorithm, which is described in the following section, was implemented in Java 6 and executed on an average specification desktop computer running an Intel(R) Core(TM) 2 Duo CPU, Model number E8400 running at 3.00GHz with 2GB of DDR2 RAM at its disposal.

4.4.1 Classification Algorithm

The classification algorithm used was taken from the Waikato Environment for Knowledge Analysis (WEKA) package [59] which is supplied as a Java library that is easily incorporated as an integral component of the system developed. After some initial empirical observations conducted using the system with a number of different classifier types, including tree and Bayesian classifiers taken from the WEKA package, a k-Nearest Neighbour Classifier was selected as the most promising of the available classification algorithms for the task. The k-nearest neighbour algorithm was used with the default parameter settings with the exception of k which after some manual tuning was set to 2.

The following Section describes the EA that was implemented including a description of the custom operators required for the variable length representation described.

Algorithm 1 Pseudo-Code describing the algorithm steps

Require: Tr = training set of 685 problems (every odd numbered problem [1 - 1369])
Require: Te = test set of 685 problems (every even numbered problem [2 - 1370])
Require: $Eval$ = evaluation set of 137 problems (every 5^{th} problem from Tr)
Require: Evo = evolution set of 548 problems s.t. $Evo \cup Eval = Tr$
Initialise population
for all $Individuals \in population$ **do**
evaluate individual*
end for
best ← getBest()
repeat
parent1 ← tournament selection()
parent2 ← tournament selection()
child ← crossover (parent1, parent2)
child ← mutate(child) : probability of 0.02
evaluate child*
best ← getBest()
until 1000 generations have elapsed
$^{*}described\,by\,Algorithm\,2$

Algorithm 2 *evaluate individual (ind)

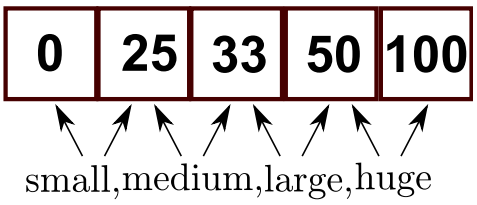
Require: classification data = $\emptyset$
for all $p \in Evo$ **do**
encode p using ind
append encoding and best heuristic for p to classifier data
end for
build classifier using classifier data
$fitness \leftarrow classification\,accuracy\,on\,Eval$

4.4.2 EA Description

After some initial observations using different parameter settings a steady state messy EA was implemented that used a population size of 40 with crossover performed to generate one offspring each iteration with a probability of 60% and mutation performed with a probability of 2%. Descriptions of the custom representation and evolutionary operators implemented are described in the following sections.

4.4.3 Representation

The chromosome representation used by the EA is derived from the approach used in [101] where a problem instances' state is described by a number of domain specific characteristics which included the percentage of each instance's items with weights within certain predetermined ranges, measured as ratios of the bin capacity.



Huge: $\frac{C}{2} < \omega_j$

Large: $\frac{C}{3} < \omega_j \leq \frac{C}{2}$

Medium: $\frac{C}{4} < \omega_j \leq \frac{C}{3}$

Small: $\omega_j < \frac{C}{4}$

Figure 4.5: For a chromosome with n genes numbered from left to right the percentage of items p_i falling into each range $r_i < p_i \leq r_{i+1}\ \forall i = 1, \ldots, n-1$ is encoded and passed to the classifier as predictor attributes. The terminal alleles, 0 & 100 were inferred.

The ranges used in [101], and adopted here as a benchmark, are shown in the chromosome representation depicted in Figure 4.5. Note that in the actual implementation the terminal allele values were inferred as they always equate to 0 and 100. These ranges were deemed ''natural" choices by the authors of [101] as at most one *Huge*, two *Large* or three *Medium* items can be placed in any individual bin. These ranges, or divisions, are used as the classifier's predictor attributes with the best heuristic being the goal, or class attribute.

An EA is used to evolve variable length chromosomes which are constrained to a maximum length that was incrementally increased for each of the experiments con-

ducted. An chromosome is used to encode each instance from the *evolution* training set by calculating the percentage of items with weights within each size range. These values are supplied to the classifier along with the known best heuristic for each instance[1].

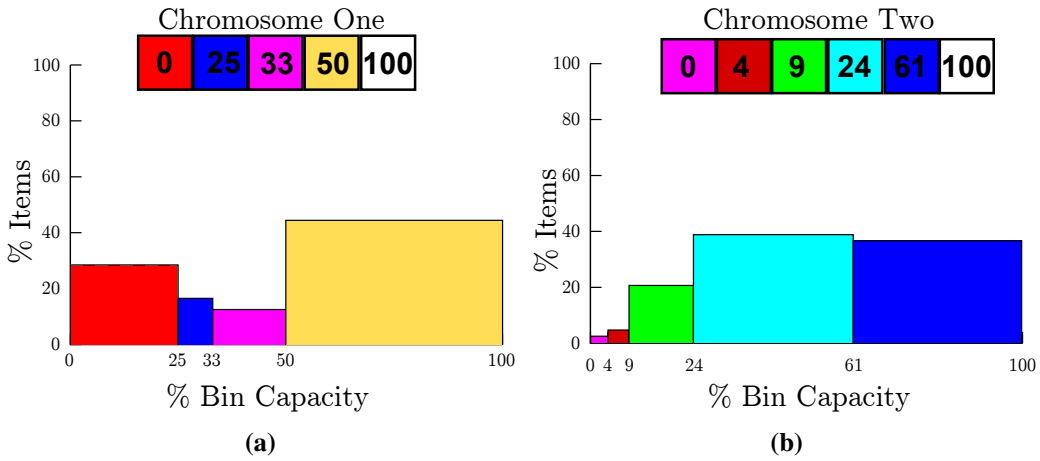


Figure 4.6: The two graphs show the *same* problem instance encoded by the two different chromosomes shown. The x-axis depicts the evolved ranges expressed as a percentage of the bin capacity whilst the y-axis depicts the percentage of the instances' items with sizes falling within each range.

Figure 4.6 shows how the same problem instance is encoded by two different chromosomes. Each member of the initial population is created by selecting the number of ranges randomly from a uniform distribution between a minimum value of 0 and a maximum value which is the same as the maximum allowed chromosome length for that experiment. The required number of boundary values are then selected at random from a uniform distribution of values between $(0, 100)$. These allele values are then sorted in ascending order from left to right in the chromosome. As mentioned the terminal values of 0 and 100 are inferred.

As an example take a single problem instance defined by $C = 100$, $n = 100$, $\omega_{1-n} = [1, 2, 3, \ldots 99, 100]$ i.e. their are exactly 100 items all with unique weights

[1]Determined using Equation 3.1 with ties awarded to the computationally simplest heuristic in the order FFD, DJD, DJT and ADJD.

ranging from 1 to 100. If encoded using chromosome 1 there would by a ratio of 25% of the problems items in the first range described by the chromosomes first two allele values incorporating items with sizes between 0% x C and 25 % x C. Only 8 % of items fall within the 25-33 range, 17% of the items would be in the range 33-50 and 50% of the problem instances items would be in the range 50-100. If the best heuristic for this instance were FFD then the information passed to the classifier for this instance would be

0.25, 0.08, 0.17, 0.50, FFD

However if the same problem instance is encoded using chromosome 2 then classifier would be presented with the tuple

0.04, 0.05, 0.15, 0.37, 0.39, FFD

In the second case 4% of items are within the first range (0 - 4), 5% are within the second range (4 - 9), 15% of the problems items fall within the third range defined by 3rd and 4th allele values (9 and 24) and 37% fall within the range 27-61. The remaining 39% of items have item sizes lying in the last range defined by 61-100.

Each of the 548 problem instance in the evolution set is characterised in this way and the subsequent list of data (one line for each instance) is supplied to the classifier. The objective is to find a set of characteristics that improve upon the ability of the classification algorithm to predict the best heuristic for each problem instance in the evaluation set. The fitness value given to a chromosome is the accuracy attained by the classifier on the unseen evaluation set of 137 problem instances.

4.4.4 Fitness Function

The ratio of *evaluation* training problems correctly classified by the classifier is used as the objective fitness value for an individual chromosome.

4.4.5 Parent Selection and Crossover

Descendants are generated by means of crossover between two parents. Each parent is selected by means of a tournament between two randomly chosen competitors with the one with the greater fitness value selected as a parent.

Crossover takes the first parent and selects all alleles up to and including a random position, placing these into the offspring. The second parent is then searched sequentially until an allele value is found greater than has been introduced from the first parent. This and subsequent genes are appended to the offspring. This process is illustrated in Figure 4.7

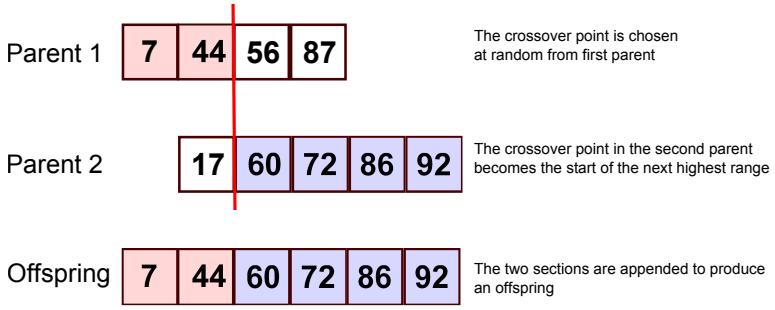


Figure 4.7: A visualisation of the crossover operator used by the EA. Mutation may alter the length of a chromosome

4.4.6 Mutation

Mutation simply adds or removes, with equal probability, one random value resulting in a mutated chromosome which varies in length by one allele than the original. Figure 4.8 depicts the addition of a randomly selected value to the chromosome.

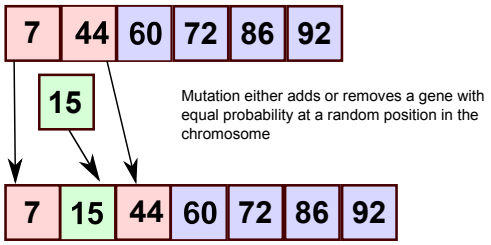


Figure 4.8: A visualisation of the mutation operator used by the EA. Mutation may alter the length of a chromosome

4.4.7 Managing Bloat

If left unattended there is a possibility that chromosome lengths would grow uncontrollably as a result of both the crossover and mutation processes employed. It was hypothesised that this would have a derogatory effect either resulting in a set of predictor attributes too large for the classification algorithm to make sense of. To manage this and to investigate the affect that the maximum allowed chromosome length exerts upon the classification algorithm a number of different limits were imposed. The results presented in Section 4.5 show the effect of varying this parameter between 7 different values ranging from a maximum length of 3 up to a maximum length of 200. Each iteration any newly created chromosomes are checked to ensure that they do not exceed the enforced limit for that experiment. The trimming process employed and described by Figure 4.9 merges the two numerically closest allele values in an oversized chromosome into one. The new allele takes on a value that is the average of the two allele values it replaces. This trimming procedure is repeated as necessary until the chromosome is at most the maximum length allowed for that experiment.

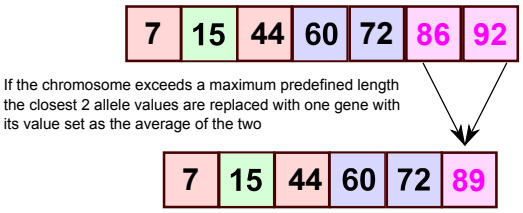


Figure 4.9: A visualisation of the bloat reducing mechanism used by the messy EA

4.4.8 Replacement and Diversity

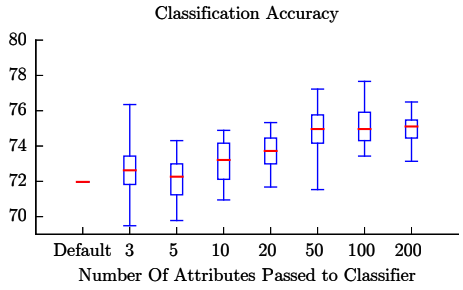
Each iteration the worst member of the population is replaced by the child if the child's fitness is better than that of the worst individual. Diversity within the population is maintained by prohibiting inclusion of any new chromosomes that are identical to any that exist in the current population.

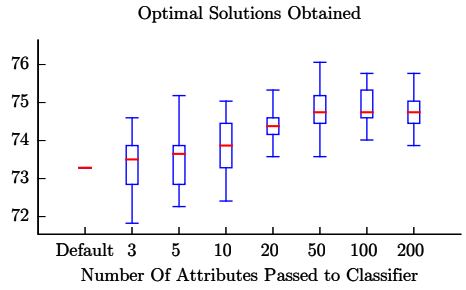
The following section describes the experiments conducted and presents the results attained using the system described in this chapter.

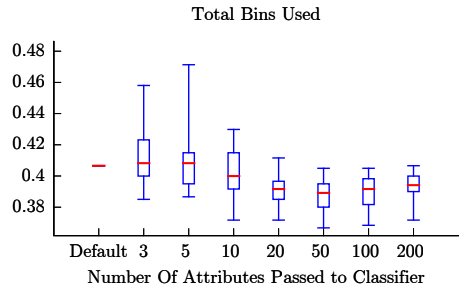
4.5 Experiments and Results

Seven experiments were conducted, each consisting of thirty runs with each run terminated after 1000 iterations. For each experiment, the only parameter modified was the maximum allowed chromosome length, $l \in \{3, 5, 10, 20, 50, 100, 200\}$. A chromosome length of l corresponds to $l + 1$ ranges once the inferred terminal alleles representing 0 & 100 were added.

The results obtained are shown in Figure 4.10. The best single individual heuristic,







	Accuracy		Solved		Bins	
Att	3	200	3	200	3	200
Mean	72.62	74.93	73.40	74.74	0.41	0.39
SD	1.32	0.83	0.75	0.45	0.018	0.008
Normal	Y	Y	Y	Y	N	Y
t-test	1.27^{-10}		6.84^{-11}			
Wilcoxon					5.86^{-06}	

Shown are the statistical test results obtained for each graph by comparing the data found for 3 and 200 attributes.

Figure 4.10: The three plots, taken over 30 runs show, for the unseen 685 test problems, the percentages of problems correctly classified and solved to the known optimal along with the percentage of extra bins over the optimal of 60257 required. The default values show the results obtained when using benchmark attributes (0.25,0.33,0.5). The results of two unpaired two tailed t-tests with no assumption of equal sample variance are given for the data sets that a Shapiro-Wilk Normality test reported as being normally distributed with a non-parametric Wilcoxon Mann-Witney test used for the other.

when ranked by the number of optimal solutions found, was DJT which solved 62.77% (430) of the instances in the test set using an extra 0.75% more bins (452) than the optimum. In comparison the hyper-heuristic presented here found 521 (76.06%) optimal solutions using only 0.37% (223) more bins.

A ten fold cross-validation was also conducted using the complete set of 1370 problems and the best set of evolved predictor attributes achieving 72.99% accuracy in comparison to 68.90% using the non-evolved default attributes.

Unlike in [101], the system described here is unable to solve any instances to the optimum that are unsolved by any of the constituent heuristics. As different heuristics, methodologies and problem instances are used a direct comparison is not entirely possible. However for comparison, when trained using the evolved characteristics that gave the best result in terms of the number of optimal solutions obtained along with the truncated training set of problems used in [101] the system presented here was able to find optimal solutions to 172 of the 223 test problems used in [101] as opposed to 166 reported by the papers authors.

4.6 Conclusions

By combining a set of diverse heuristics and exploiting their strengths on the areas of the problem space that they perform best on their inherent weaknesses can be partially overcome. It is well known that no heuristic is able to perform consistently well across the full problem space and the study shows that this inherent weakness in individual heuristics can be minimised. The study presented in this chapter shows that by combining the abilities of even a small number of deterministic heuristics for the BPP that the quality of the solutions obtained when applied to a large diverse set of problem instances is improved significantly over those attained using any individual heuristic. Furthermore by evolving *relevant* predictor attributes for use by the classifier the goal of generating a problem description that maps individual instances to an appropriate heuristic for solving it was improved. The hyper-heuristic developed is able to better generalise over a wider range of problem instances with varying characteristics than can be addressed by any of the heuristics when used in isolation. The new heuristic introduced previously, ADJD, has been shown to perform better on problem instances with certain characteristics than any of the other heuristics investigated and although it is the single worst heuristic when evaluated over the complete set of benchmark instances it is shown to increase the generality of the hyper-heuristic system presented by a significant margin.

Initial investigations into increasing the number of heuristics used suggests that the classification task increases steeply in complexity with the number of heuristics and although there is potential for increasing the quality of the solutions attained this is limited by the small number and lack of diversity of deterministic constructive heuristics for the off-line 1D BPP in the literature. Other deterministic heuristics were investigated, such as BFD and SS, but were deemed too similar or unproductive rarely improving on the solutions attained by the combination of heuristics used.

The primary observations and conclusions drawn from the study which influenced the remainder of this thesis are:

- None of the heuristics dominates any other when evaluated across the complete set of problem instances investigated.
- Individual heuristics do outperform others on subsets of problems.
- Heuristics that perform well on problem instances generated using a particular combination of parameters are more likely to perform well on unseen problem instances generated from the same set of parameters.
- The problem instances that a heuristic will work well on can be predicted using classification techniques.
- The classification algorithm's accuracy can be increased by evolving, rather than manually designing, the predictor attributes used.
- Selecting the best heuristic, for each of a wide variety of problem instances, from a set of diverse heuristics can significantly improve the overall solution quality when compared to the solutions obtained using any single heuristic.

The study presented in this chapter uses deterministic heuristics and static problem instances. As mentioned at the start of the chapter it would be trivial to apply each heuristic in turn and greedily select the best one for each problem instance. The results presented in this chapter can be at best only as good as the results obtained using a greedy selection strategy. While it was the intention to show here that it was possible to predict which heuristic would be chosen if a greedy selection strategy was employed

the technique is ultimately limited by the collective ability of the set of heuristics used. The previous chapter showed that many human-designed heuristic perform similarly on many problem instances which from the perspective of a selective hyper-heuristic limits the potential of any approach. In order to alleviate this limitation the remainder of this thesis concentrates on generative hyper-heuristics which attempt to automate the heuristic design process and remove the limitations imposed human designers. The following chapter introduces a generative hyper-heuristic that is used to generate individual heuristics that are evaluated against large problem sets. This is expanded in subsequent chapters to consider the observation made here that having sets of heuristics that individually work well on different niche areas of the problem space significantly improves the potential utility of a selective hyper-heuristic approach.

Chapter 5

Generative Hyper-Heuristics

In the previous chapter the utility of using a selective hyper-heuristic to select from a pool of predetermined simple deterministic human designed constructive heuristics for the 1D BPP was explored. The heuristics selected from the literature are limited by the domain knowledge and the imagination of the human designer and whilst individual heuristics are suited to certain problem instances, often different heuristics were shown to produce similar or identical solutions for many of the benchmark problem instances that they were evaluated on. This chapter looks at the second major class of hyper-heuristics; heuristics to generate heuristics or generative hyper-heuristics where a Genetic Programming technique is utilised to automate the heuristic design process in an attempt to improve upon the deterministic heuristics created by human algorithm designers.

5.1 Contribution

This chapter introduces a generative hyper-heuristic that is used to generate heuristics for the BPP that are shown capable of outperforming a range of well researched deterministic constructive heuristics on a large diverse set of problem instances. Unlike other generative approaches from the hyper-heuristic literature GP is not used to create heuristics that explicitly decide where to pack each item based on the current state of the solution. A compact form of GP, called Single Node Genetic Programming

(SNGP) [64, 65] is employed to generate combinations of constituent elements that are evaluated by measuring the *side effect* that they cause when applied to a problem. The process of executing the evolved heuristic directly causes items to be placed into bins rather than the more conventional approach of using a wrapper to decode the output of the evolved program in order to decide on the placement of items. The heuristics evolved are shown to outperform 4 human designed heuristics sourced from the literature on a large set of 1370 problems, 685 of which are unseen during the evolution stage.

5.2 Introduction

Generative hyper-heuristics are a class of autonomous algorithm that aim to automate the heuristic design process.. In contrast to selective hyper-heuristics, generative approaches do not search over a set of pre-defined heuristics but search over a set of *components* from which novel heuristics can be fabricated. Generative hyper-heuristics can be sub-classified [19] based on the class of heuristics that are generated; either perturbative or constructive and also by the use or omission of memory / learning mechanisms. The approach here focuses on generating deterministic constructive heuristics for application to the off-line version of the 1D BPP. Memory is encapsualated during an off-line learning strategy with the resultant heuristics tested against a large corpus of previously unseen problem instances.

5.3 Background and Motivation

GP can be applied to many different types of problem, in theory fully functional programs could be evolved. In the seminal work of Koza [72] four simple introductory examples are given where GP is used to evolve programs for control, planning, symbolic regression and function approximation. All of these implementations use a wrapper to decode the output returned by evaluating an evolved tree structure before translating this output to an appropriate action. All of the generative hyper-heuristics reviewed in Section 2.3.3 use the same approach where a wrapper is employed to determine the

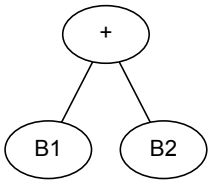


Figure 5.1: Example GP tree that causes a side effect: During the course of evaluating the tree nodes B1 and B2 are evaluated. B1 causes a side effect where the largest item possible is packed into the current bin. Similarly B2 causes the largest possible 2 items to be packed into the current bin .

action to take based on the result of evaluating an evolved function which encapsulates information about the problem state, such as item length and bin capacity. As an example, Figure 2.3 (shown in Section 2.3.3) shows how the best fit heuristic could be implemented using the nodes described in [9]. In this example the tree is evaluated in its entirety for each individual item in a problem instance. The result obtained is passed to an encompassing wrapper which is used to decode the output and to decide which of two possible actions to take; pack the item into the current bin or open a new bin and place the item there.

A second GP implementation is also described by Koza [72] where evaluating the tree structure can cause a *side effect* in the environment. Actions are completed explicitly by individual nodes in the tree rather than by an encompassing wrapper. Figure 5.1 shows a simple example of a tree structure that includes 2 of the nodes described in detail later in this chapter ($B1$ and $B2$). When evaluated these nodes may cause a *side effect* where the biggest possible 1 or 2 items (B1 and B2 respectively) are immediately placed into the current bin. The code to perform the packing task is not placed into an encompassing wrapper but is included within the node. Items are packed immediately without the requirement to completely evaluate the tree structure and decode the result. A wrapper is still employed to make other decisions such as the opening and closing of bins but the packing of items is explicitly completed by evaluating single nodes in the tree. This process is described in more detail later in the chapter.

As mentioned previously, hyper-heuristic approaches have highlighted the utility of combining simple heuristics to improve on solution quality. In the realm of bin

packing combinations of heuristics, applied to pack different bins from the same problem instance, have been shown in some cases to outperform the constituent heuristics [78, 101]. This ideology dates back to Fisher and Thompson [47] who used machine learning techniques to select combinations of simple heuristics (dispatching rules) to produce solutions to local job-shop scheduling problems, a technique that has inspired many more recent publications [60]. In previous chapters 4 simple heuristics have been investigated; FFD, DJD, DJT and ADJD. With the exception of FFD all of these heuristics use a similar strategy (and hence they often produce similar solutions). They first simplify the computationally expensive problem of finding a combination of items that fully fill an empty bin by partially filling that bin with a few items before conducting a simplified search to find a set of items to fill the remaining space. FFD simply packs each item, taken in descending order of size, into the first available bin that will accommodate it. A motivating factor for the research presented in this chapter is that with the exception of FFD it would be possible to use the other heuristics in different combinations to pack individual bins, possibly improving solution quality. However as the heuristics are similar initial studies showed that there was little benefit using this approach with the limited set of human designed heuristics available. The remainder of this chapter takes this concept a step further by designing a set of components (nodes), based on the 4 heuristics mentioned that can be combined to produce new heuristics. Before describing the set of nodes devised to construct new novel heuristics, SNGP and the motivation for its used are described.

5.4 Single Node Genetic Programming

Traditional GP employs a population of tree structures, which represent computer programs, that are acted upon by evolutionary operators such as crossover and mutation to search for improved solutions to a variety of problems. Unlike many evolutionary techniques where the size of the representation is fixed, GP suffers from the affect of *bloat* [77] where the tree structures can grow undesirably large through the process of combining different branches from different trees in the population. For the problem and representation investigated in this chapter this affect was found to be highly in-

efficient. As explained later, the programs evolved here are used repeatedly to pack individual bins. Evaluation of a single node can cause up to 5 items to be packed into a bin. Given the distribution of item weights for the benchmark problem instances used to evaluate the approach, creating tree structures that grow in size is unproductive as the only the first few nodes evaluated will manage to place items into a bin. An encompassing wrapper detects the failure to pack any more items and causes the process to repeat using a new bin.

Initial observations using conventional Koza style GP highlighted the detrimental affect of bloat where the tree structures evolved became ever larger with many nodes proving redundant. Based on this observation and the relatively coarse grained approach to the design of the nodes (explained in the following section), a more compact variation of GP was adopted. Single Node Genetic Programming (SNGP), introduced by Jackson in [64], eliminates bloat due to its use of a single mutation operator. With no crossover employed there is no mechanism to allow the structures generated to grow beyond their initial size. SNGP also allows for more complex programs (using the same number of nodes) to emerge than would be possible using GP which restricts the topology of programs to tree structures.

Originally applied to 3 problems amenable to being solved using dynamic programming, namely 6 multiplexer, even-parity and symbolic regression[64], SNGP was further investigated in [65] where its effectiveness on problems where the solution is obtained as a *side effect* was explored. Three such problems were tackled using SNGP; the Santa Fe artificial ant problem, a maze navigation task and third problem where the objective was to generate a program capable of parsing arithmetic and logical expressions. SNGP was shown to be an effective approach for tackling this class of problem showing significant improvement to results obtained using conventional GP.

SNGP differs from the conventional GP model introduced by Koza [72] in a number of key respects.

- Each individual node in the network may be the starting point for evaluation, not only the top most node.
- Nodes may have any number of parent nodes (including none and duplicates) allowing for network structures other than *trees* to be formed.

- The only evolutionary operator used is mutation which is employed as a hill climber with the mutation undone if no improvement is achieved.

Figure 5.2 shows two partial SNGP structures with any nodes not connected to the current top node omitted for clarity. The standard tree structure on the left show how the DJD heuristic could be represented using the nodes outlined in Table 5.1. The right side of the diagram highlights a key difference between SNGP and conventional GP; that individual nodes are permitted to have multiple parent nodes. In keeping with the

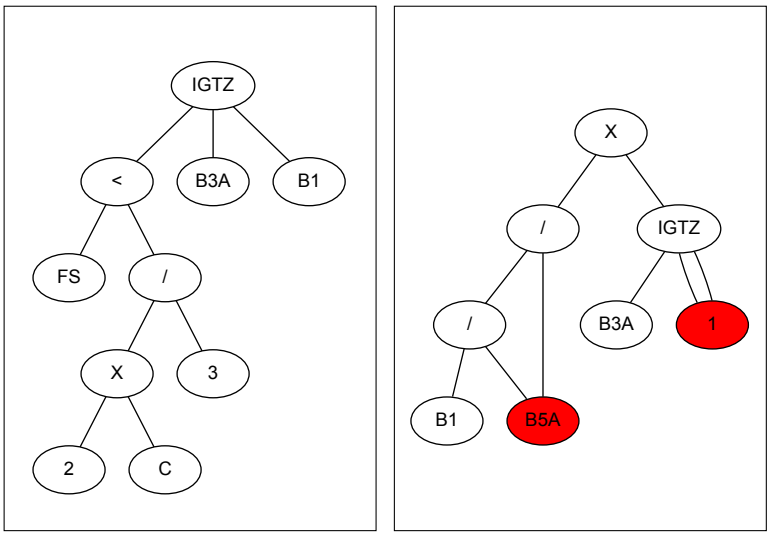


Figure 5.2: Two example SNGP structures. The tree on the left shows how the DJD heuristic could be formulated whereas the tree on the right highlight a key difference between SNGP and conventional GP in that nodes may have more than 1 parent node. There may exist unconnected nodes which in both diagrams are removed for clarity.

terminology used in the original literature each node is considered as an individual in a population. Each node can be the starting point for evaluation making each a unique heuristic. It should be noted however that the only evolutionary operator used is mutation which is applied to the complete network structure and that the concept of considering nodes as individual entities in a population seems to have little in common with other population based approaches.

Figure 5.3 depicts how each node in an SNGP network structure is treated as a distinct heuristic by considering each node and its connected children in isolation.

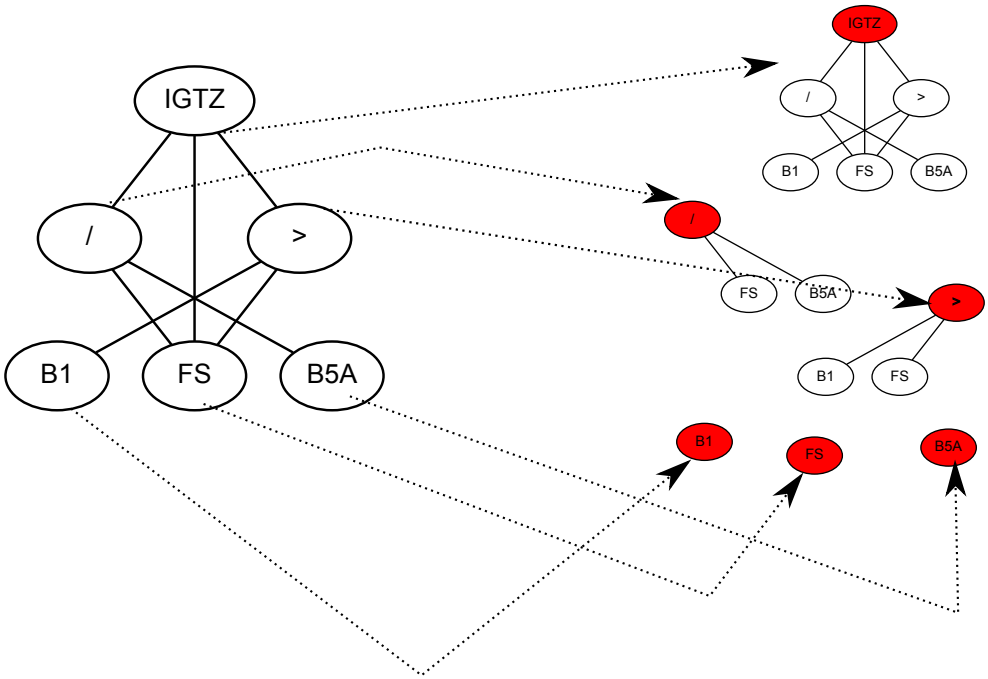


Figure 5.3: An SNGP structure comprising of 6 nodes (left) decomposed to create 6 distinct heuristics (right). Each node in the structure may be used as the starting point for evaluation and is effectively a different heuristic

The initialisation process for SNGP is covered by the first 3 steps in the following numbered list with steps 4 and 5 constituting the main evolutionary loop that is executed repeatedly until some predefined stopping criteria is met.

1. Each terminal node $T \in \{t_1, \ldots t_r\}$ is added once and given an integer identification number ranging from $1 \ldots r$.

2. A number, n, of function nodes are selected at random from the set of all function nodes $F \in \{f_1, \ldots, f_s\}$ and given an identification number ranging from $r + 1$, $\ldots r+n$. Function nodes may be duplicated or omitted from the SNGP structure.

3. Each function node has its child nodes assigned at random from the set of all nodes with a lower id to prevent any recursive loops.

4. A single mutation operator is used which selects a function node at random and

reassign one of its edges to point at a node chosen randomly from the set of all nodes with a lower identification number.

5. If no node from the new mutated network proves more effective on the training problems then the network is reverted to its previous state.

The SNGP structure is initialised with exactly 1 of each of the available terminal nodes and a predefined number of randomly selected function nodes (this may omit some function nodes or introduce duplicates). Connections between function nodes and child nodes are set randomly while observing the constraints that each function node must have all of its child nodes assigned and that no recursive loops are introduced. This procedure results in a network where terminal nodes may have no parents and where different portions of the overall network structure are disconnected. The ability to use any node in the structure as the root of the evolved program still holds although evaluating some nodes may cause no effect. A wrapper is used to control the system response to such events and prevent infinite looping. During the mutation process a connection currently existing between two nodes is randomly selected and reassigned. When re-evaluating a node only it and those nodes connected recursively as child nodes need to be considered, making the procedure more efficient. Each node receives a fitness value as if it were an individual in the population. Evaluation of the population, used to drive evolution, may either be carried out by averaging the fitness of all nodes or by considering the node with the best fitness in isolation. The second elitist measure is adopted in the application of SNGP described over the remainder of this chapter.

5.5 Implementation

In the following section the set of component function and terminal nodes that were implemented after analysing and decomposing four human designed heuristics; FFD, DJD DJT and ADJD are described.

5.5.1 Nodes

As previously noted, all of these heuristics with the exception of FFD can be used to pack a single bin in isolation at which point the bin can be closed and the procedure repeated until all items are packed. FFD, in its original implementation requires that all bins remain open for the duration of the packing procedure. It was noted that FFD could be rewritten to pack a single bin at a time by iteratively packing the next largest item from the set of remaining unpacked items that would fit into the current bin. Once no more items were able to be placed into the bin it could be closed and the procedure repeated. Although this process is more computationally expensive, requiring multiple searches through the set of remaining items, it allows the FFD heuristic to be used in the same manner as the other heuristics investigated. The other heuristics all use similar strategies to minimise the computational complexity associated with searching for combinations of items whose combined weights best fill the space in a bin. When a new bin is opened an initial filling stage partially fills the bin before a computationally expensive partial search attempts to optimise the filling of the remaining free space. DJD and DJT both operate identically during the initial filling stage by iteratively packing the largest available items into a bin until that bin is at least one third full. ADJD differs in that it continues to pack the largest available items until the free space is at most three times the size of the average weight of the remaining items. After the initial filling stage both DJD and ADJD search for a combination of up to 3 items to fill the remaining space in a bin whereas DJT considers combinations of up to 5 items. Based upon this knowledge the nodes described in Table 5.1 were implemented. All of the human designed heuristics mentioned previously can be created using different combinations of these nodes in conjunction with the wrapper described later. As an example the DJD heuristic is shown in Figure 5.2. The motivation behind combining simple components using SNGP lies in the hypothesis that the automated process would be able to find effective combinations of these constituent parts that a human designer, limited by preconceptions of how to tackle a particular problem would not envisage. The set of terminal nodes contains both nodes that have a direct effect on the solution (a node can place between 1 and 5 items into a bin) and nodes that return values representing the state of the current bin such as capacity C and free space FS.

A wrapper, described in the following section, encompasses the SNGP structure and is used to decide when to open a new bin or to terminate the packing procedure.

Table 5.1: Function and terminal node descriptions

Function Nodes	
/	Protected divide. Returns -1 if the denominator is 0 otherwise the result of dividing the first operand by the second is returned
$>$	Returns 1 if the first operand is greater than the second or -1 otherwise
IGTZ	Is Greater Than Zero: If the first operand evaluates as greater than zero then the result of evaluating the second operand is returned. Otherwise the result of evaluating the third operand is returned
$<$	Returns 1 if the first operand is less than the second or -1 otherwise
X	Returns the product of two operands
Terminal Nodes	
B1	Packs the single largest item into the current bin returning 1 if successful or -1 otherwise
B2	Packs the largest combination of exactly 2 items into the current bin returning 1 if successful or -1 otherwise
B2A	Packs the largest combination of up to 2 items into the current bin giving preference to sets of lower cardinality. Returns 1 if successful or -1 otherwise
B3A	As for B2A but considers sets of up to 3 items
B5A	As for B2A but considers sets of up to 5 items
C	Returns the bin capacity
FS	Returns the free space in the current bin
INT	Returns a constant integer value randomly initialised from $[-1, 1, 2, 3, 4, 5]$
W1	Packs the smallest item into the current bin returning 1 if successful or -1 otherwise

5.5.2 SNGP Wrapper

Although the process of packing items into a bin is conducted by explicitly by evaluating certain terminal nodes the implementation still requires the use of a wrapper to oversee the execution of each node.

Not all of the nodes generate a side effect of packing items into the current bin when evaluated. Whilst function nodes will always have their child nodes assigned and are therefore never considered in isolation, nodes such as C and FS if evaluated in isolation cause no effect and their repeated execution would result in an infinite loop

where no items would be placed into the solution. In order to ensure that all heuristics terminate, even when no items are packed, a wrapper, described by Algorithm 3, is used to encompass the node being evaluated. The wrapper is responsible for determining when to open a new bin based on both the value returned by the heuristic under scrutiny and the changing state of the solution currently being constructed. If the node under evaluation returns a value of < 0 a new bin is opened. If it returns a non negative value and there are still items to pack and at least one item was packed during the last evaluation the node is evaluated again. If the node packs no items and there are still items remaining to be packed each item is placed into its own bin and the process terminates. This causes terminal nodes that do not cause a side effect to generate a solution using one bin for each item in the problem instance which consequently causes the node to receive the worst possible fitness score causing the node to be subsequently disregarded by the evolutionary process.

Algorithm 3 SNGP Node Wrapper

Require: $I \in \{i_1, i_2, ..., i_n\}$ {The set of items to be packed}
Require: $B = \emptyset$ {The set of bins which is initially empty}
  repeat
    add a new bin b to B
    repeat
      $I' = I$
      $result = evaluate(Node)$ {This may cause items from I to be packed into the current bin b}
    until $result < 0$ **or** $I = \emptyset$ **or** $I = I'$
    if $I = I'$ **and** $I \neq \emptyset$ **then**
      pack each remaining item in a new bin
    end if
  until $I = \emptyset$

5.6 Experiments and Results

The generative hyper-heuristic described in this chapter was trained and tested on equal divisions of the 1,370 benchmark problem instances from Problem Set A, taken from the literature and summarised in Section 3.3.6. The division of Problem set A into equal sized training and test sets was as described in the previous chapter. This ensures an equal distribution of problem instances generated using the same parameters with every second problem instance placed into the test set. During training the objective was to evolve a single heuristic that minimises the total number of bins used when applied to the complete training set. This value is assigned as the fitness value for each node that constitutes the SNGP network.

The hyper-heuristic was trained for 500 iterations on the training set before the system was halted and the single best heuristic was evaluated on the training set. The only other variable parameter required was to define the number of terminal nodes used to initialise the system which was fixed at 12 after initial experimentation.

The System was executed 30 times from a clean start in order to gain statistically relevant results with each run terminated after 500 iterations. Only the first 250 generations are reported here as no improvement was observed in any of the 30 runs after this point. The SNGP structures were initialised as described in Section 5.4 using 12 randomly selected function nodes. The software was implemented in Java and executed on a high performance cluster comprising of 18 servers each equipped with dual, quad-core cpu's with 16Gb ram running Fedora 12.

The results are summarised in Table 5.2 which shows for comparison the number of problem instances that were solved optimally by each of six deterministic heuristics described in Section 3.4. The number of extra bins more than the optimal of 60257 required by each heuristic when applied to the same test set of instances is also given. The table also presents the results obtained by treating each of the terminal nodes that are capable of packing items as an individual heuristic. This highlights the benefits of using SNGP to evolve heuristics which are composed of combinations of nodes used in the terminal set. It is interesting to note that by encompassing each of the terminal nodes in a *wrapper* that continues to execute until no more items are packed into a bin that many of the simple terminal nodes when used in isolation can outperform some of

the human designed heuristics. For example $B2A$ finds more optimal solutions than the best human designed heuristic if gauged by this metric. In apparent contradiction to this nodes such as $B3A$ which performs poorly in terms of the number of optimal solutions found perform relatively well when gauged by the number of extra bins over the optimal number required. It is clear that the evolved heuristic outperforms the rest by a substantial margin regardless of the qualitative measure used.

Table 5.2: Comparison between 4 deterministic heuristics, the terminal nodes that cause a *side effect* and the best generated heuristic (HGEN) on the test set of 685 problem instances.

Heuristic	Optimal Solutions Found	Extra Bins Required
FFD	393	1088
DJD	356	1216
DJT	**430**	**451**
ADJD	336	679
BFD	394	1087
SS	383	1112
Terminal Node	Optimal Solutions Found	Extra Bins Required
B1	393	1088
B2	308	3250
B2A	**432**	944
B3A	303	764
B5A	332	**692**
W1	31	16761
Generated Heuristic	Optimal Solutions Found	Extra Bins Required
HGEN	**518**	**257**

Figures 5.4 and 5.5 show results for single heuristic generation taken over 30 runs. The box plots illustrate how 25 of the 30 runs evolve in less than 250 generations to give the same results using both the number of instances solved and the total number of bins required as metrics. For those 5 runs where the results deviated from the best the deviation was minimal. Using a Mann-Whitney rank sum test to compare the best

human designed heuristic (DJT) to the best evolved heuristic using the number of bins required as a metric offers little insight into the obvious improvement in the results due to the fact that both heuristics generate solutions which require an equal number of bins for many of the problem instances. However if Falkenauer's fitness function [40] is used as a metric then the one-tailed and two-tailed P values obtained show the results to be highly significant measuring at 5.48×10^{-5} and 10.96×10^{-5} respectively. Two of the best heuristics generated during two independent runs, randomly selected from the 30 evaluations conducted are shown in Figure 5.6.

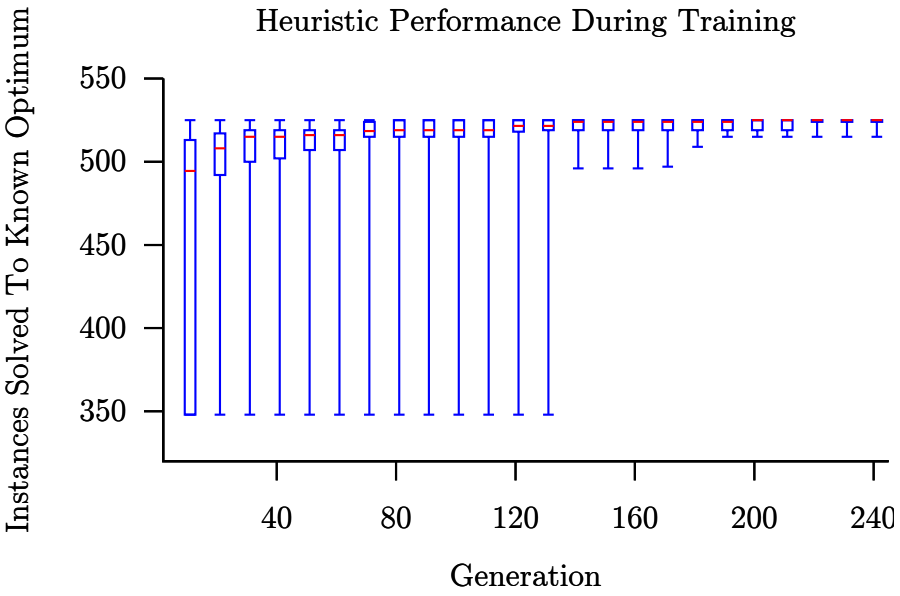


Figure 5.4: Heuristic performance over 30 runs using the quantity of the 685 training problem instances solved using the known optimal number of bins as a metric.

Table 5.3 gives the number of problems solved using the specified number of extra bins than the known optimal when the best heuristic evolved is applied to all of the 1370 problems from problem set A. The results obtained by each of 4 benchmark heuristics are also shown for comparison. The results obtained by the best single evolved heuristic on the training, test and complete set of problems from Problem Set A are summarised Table 5.4. The table also shows the results obtained by the best heuristic on each of these problem divisions which in all cases was DJT.

In order to provide a further comparison, the best heuristics obtained from each

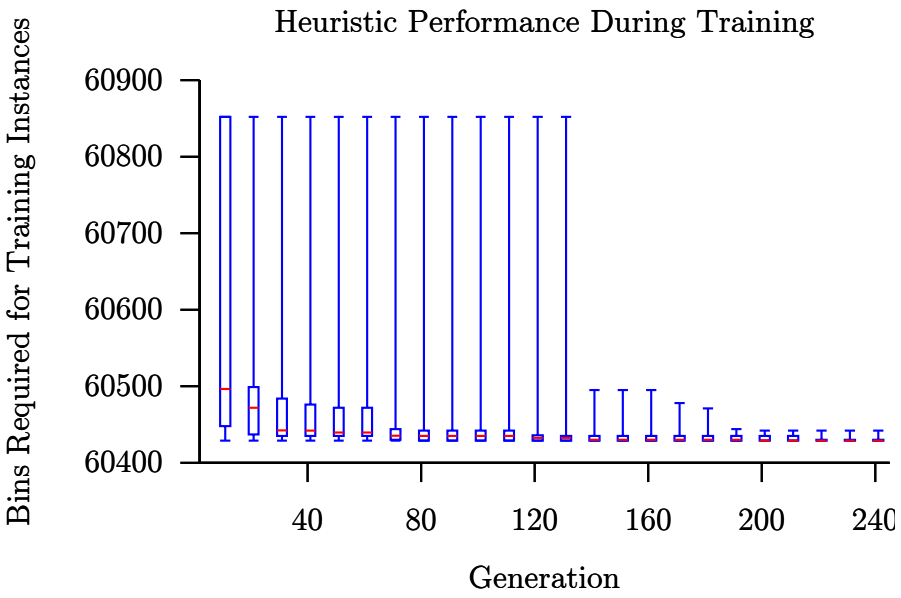


Figure 5.5: Heuristic performance over 30 runs using the total number of bins required to pack all 685 training instances as a metric

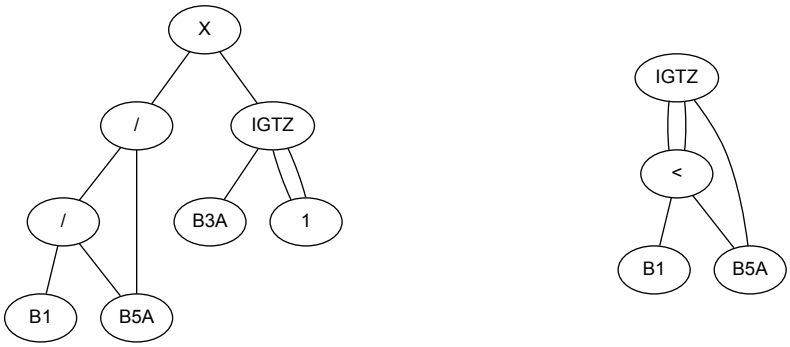


Figure 5.6: Two of the *best* heuristics generated during two independent runs, randomly selected from the 30 evaluations conducted are shown.

run were used to solve the much larger set of 15830 problem instances in problem set C. The best single evolved heuristic used 7.8% fewer extra bins than were required by ADJD which was the best human designed heuristic on these problems. ADJD used 18541 extra bins that the known optimal of $1,362,542$. Table 5.5 replicates the results presented in Table 5.3 for the problem instances in problem set C. The evolved heuristic uses 7452 fewer bins to solve all 15830 problem instances than the best human

Table 5.3: Number of problems solved requiring δ extra bins on problem set A. A comparison between the benchmark heuristics and the best evolved heuristic

	Number of Problems Solved Requiring δ extra bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
FFD	788	267	78	83	39	16	18	9	18	4	50
DJD	716	281	119	58	48	36	10	16	23	3	60
DJT	863	331	90	26	30	15	11	2	1	1	0
ADJD	686	368	153	76	38	22	12	9	1	5	0
HGEN	**1028**	242	43	32	12	7	2	2	1	1	0

Table 5.4: Results Summary: Comparing the best human designed heuristic (DJT) with the best evolved heuristic (HGEN) on problem set A.

	Optimal Solutions		Extra Bins	
Problem Set	Human Best	Hgen	Human Best	Hgen
Training	433	**513**	430	**284**
Test	430	**515**	451	**266**
Problem Set A	863	**1028**	881	**550**

designed heuristic (ADJD) and finds optimal solutions in 1634 cases where none of the deterministic heuristics could solve any of the problem instances optimally. However it should be noted that ADJD solves 14031 instances using no more than 1 extra bin in comparison to 12334 for the best evolved heuristic. ADJD also solves all problem instances using at most 9 extra bins whereas the best evolved heuristic requires 10 or more extra bins in 404 cases.

Table 5.5: Problems solved and extra bins over the optimal number required on problem set C using the best evolved heuristic

	Number of Problems Solved Requiring δ extra bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
FFD	0	8887	2819	1158	1262	365	148	214	405	118	454
DJD	0	7594	2390	1316	1029	581	581	329	282	233	1495
DJT	0	10844	2188	824	577	387	318	199	59	30	404
ADJD	0	14031	1299	228	175	69	18	6	3	1	0
HGEN	1634	10700	1437	559	493	202	177	135	59	30	404

5.7 Conclusions

Single heuristics were generated during a training phase that outperformed a selection of well researched deterministic heuristics when applied to a large set of problem instances totalling 1370 problem instances split into equal sized training and test sets. The approach is novel in its use of SNGP to generate new constructive heuristics for the BPP from simple constituent elements. It has been shown that the best evolved heuristic outperforms human designed heuristics when applied to the problem instances in Problem Set A using both the number of problems solved and the total number of bins required as quality metrics.

However, a brief analysis of the solutions attained by the best evolved heuristic on the much larger and more constrained set of problems in Problem Set C shows that the evolved heuristic is outperformed in terms of the number of extra bins required on Problem set C. Whilst DJT proves the best human designed heuristic on problem Set A, ADJD is superior when evaluated by this metric on Problem Set C highlighting the effectiveness of a more dynamic packing strategy when compared to the other less adaptive human designed heuristics across extremely large unseen problem sets. (ADJD adjusts the packing strategy depending on the average item weight to bin capacity ratio).

ADJD generalises well across this larger set solving the 15830 problem instances in Problem Set C using $18,541$ extra bins than the known optimal of $1,362,542$. The evolved heuristic that proved best on Problem Set A needed $25,993$ extra bins (over

40% more.) In contrast the best evolved heuristic found optimal solutions to 1634 problem instances from Problem Set C when ADJD did not find any. This reinforces the claim that no heuristic can be successful over increasingly large and varied sets of problems and indicates that heuristics that generalise well over large portions of the search space prove less effective when contrasted to heuristics tailored to niche areas of the problem landscape. The literature on using GP to generate heuristics for the BPP is predominately focused on using GP to generate *disposable* heuristics, usually perturbative, that are tailored towards solving individual problems.

There is clearly a trade off between the ability of a single heuristic to generalise across vast data sets and its utility when contrasted to more specialised approaches tailored towards individual problem instances, or very small sets of similar problems. While the best generated heuristic clearly outperforms the human designed benchmark heuristics when evaluated individually, the collective of human designed heuristics improves on the performance of the generated heuristic when greedily applied to the complete set of problems from Problem Set A. In the following chapter this knowledge is exploited by generating *sets of heuristics* that collectively generalise across large problem sets but are individually disparate; tailored to their own niche areas of the problem landscape.

Chapter 6

Generating Sets of Co-operative Heuristics using a Genetic Programming Island Model

In the previous chapter the utility of using Genetic Programming for generating deterministic constructive heuristics for the BPP was shown. Single heuristics, evolved using Single Node Genetic Programming (SNGP), were shown to outperform 4 heuristics sourced from the literature when applied to a large diverse set of problem instances. In Chapter 4 it was shown that by exploiting the strengths of individual human-designed heuristics that that their combined ability outperforms their individual capabilities. The heuristic(s) evolved in the previous chapter were shown to be able to generalise over a wide range of problem instances with *good* performance when compared to the heuristics sourced from the literature. However the performance of any individual heuristic when designed as an average all round performer across a wide range of problem instances must be compromised when compared to more specialised heuristics tailored towards solving problem problem instances with specific characteristics. Chapter 4 showed that different human-designed heuristics performed best on niche areas of the problem space but also highlighted the similarity between the solutions attained using many human designed approaches.

This chapter expands on the system developed in the previous chapter by extend-

ing the system to generate sets of heuristics that are individually tailored towards niche areas of the problem space whilst collectively generalising over a wider range of problem instances. This is accomplished by adding an Island Model to evolve multiple heuristics in isolation that are evaluated by their collective ability on the set of problems presented. The system is trained and tested on the same divisions of Problem Set A used in the previous chapter and further evaluated on the much larger set of problems from Problem Set C. Results show that the collective ability of the evolved set of heuristics is superior to both the quality of solutions that can be attained by any single heuristic, either generated or human designed, or to the best solutions produced by the combined set of the deterministic heuristics sourced from the literature that are used throughout this thesis.

6.1 Contribution

The chapter introduces a generative hyper-heuristic that employs an island model to concurrently generate multiple heuristics using Single Node Genetic Programming. The system developed generates an undetermined number of heuristics with islands added and removed dynamically. The set of heuristics created are evaluated during training based on their collective ability to solve half of a large set of one dimensional bin packing problems containing 1370 problem instances sourced from the literature. Once trained the set of heuristics evolved are shown to collectively[1] outperform individual heuristics, generated and human designed, when applied to the 685 unseen benchmark test instances and a further 15,830 newly generated problem instances. The work contained in this chapter was originally published in [113].

6.2 Introduction

Following on from the previous chapter where Single Node Genetic Programming was used to generate single heuristics for the One Dimensional Bin Packing Problem, an island model [96] is adapted to use multiple SNGP implementations to generate diverse

[1]The best evolved heuristic is selected using a greedy selection strategy.

sets of heuristics which collectively outperform any of the single heuristics when used in isolation. The set of novel heuristics generated implicitly interact with each other using a form of cooperative co-evolution to collectively minimise the number of bins used across a large set of problem instances. The system is trained and tested on the same equal divisions of the 1,370 benchmark problem instances from Problem Set A that were used in the previous chapter. A further evaluation is conducted by applying the best set of evolved heuristics to 15,830 problems from problem set C introduced in Section 3.3.7.

Results show that the collection of heuristics evolved by cooperative co-evolution outperforms any single heuristic, adding further weight to existing evidence in the hyper heuristic literature of the utility of combining simple heuristics and the benefits of automating the heuristic design process.

The system developed is explained in the following section.

6.3 Island Model

The system described here uses multiple instances of the generative hyper-heuristic described in the previous chapter configured as an island model. Each island introduced into the system is identical in operation to the system described previously with the exception of the evaluation process which assigns fitness based on an islands ability to cooperate with the other islands in the system. The island model implemented is adapted from [96] where the authors used a novel approach for evolving "interacting coadapted subcomponents". The authors distinguish their model from other approaches, such as Learning Classifier systems which the authors describe as *competitive* rather than *cooperative*. The model is evaluated on a simple bit string pattern matching task where the number of patterns is not known *a priori* before being applied to the more complex task of evolving weights for a cascading neural network. Islands are removed if their contribution is deemed negligible and are added when the fitness of the system stagnates making the number of islands a dynamic self-adjusting property of the system.

The system implemented here evolves a set of *complementary* heuristics which

cover different portions of the heuristic search space, collectively outperforming any of the individual constituent heuristics. Figure 6.1 illustrates the concept. Non-overlapping areas of the Venn diagram describe those instances for which a heuristic uses fewer bins than any other. Heuristic H2 gives no contribution and could be eliminated as it is fully enclosed by H1. The island model described in [96] was implemented to use multiple

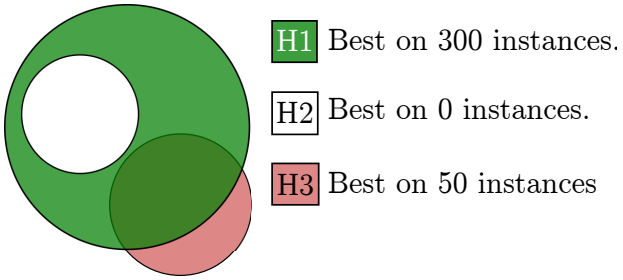


Figure 6.1: The Venn diagram conceptualises how a set of heuristics collectively improve upon their individual abilities to solve a set of problem instances. Heuristic 2 is encompassed by Heuristic 1 showing that it adds nothing to the collective performance of the set of heuristics. In contrast H1 and H2 have non-overlapping areas depicting that they are able to each find solutions on some problem instances that are better than are attained by any of the other heuristics

instances of SNGP rather than GAs. Each node in an island's SNGP network structure is evaluated by measuring its ability to cooperate with the best nodes taken from each of the the other islands. The process of co-evolving heuristics is described by Algorithm 4 and conceptualised by Figure 6.2. Note that only partial SNGP structures are depicted due to space restrictions.

The fitness value attributed to a heuristic (node) is designed to reflect its ability to cooperate with the best nodes from each of the other islands.

Fitness is calculated using Equations 6.1, 6.2 and 6.3 and is simply the sum of the number of bins fewer required by the heuristic in comparison to the best result achieved by any of the other islands best heuristics. Only problem instances where the heuristic being evaluated uses less bins than any of the other islands best heuristics are used for

Algorithm 4 Island Model Pseudo-Code

```
add one Island to the empty set of Islands
bestBins = Integer.MaxValue
repeat
  if bestBins is unimproved for 20 generations then
    for all Island ∈ Islands do
      Remove Island
      if total number of bins > bestBins then
        reinstate Island
      end if
    end for
    add new island
    evaluate all nodes in new island
    set all of the new islands nodes fitnesses
  end if
  bestBins = evaluateBins()
  for all Island ∈ Islands do
    mutate(island)
    evaluate all nodes in mutated island
    set all nodes fitnesses in mutated island
    if evaluateBins() ≥ bestBins then
      revert Island to previous state before mutation
    else
      bestBins = evaluateBins()
    end if
  end for
until 500 generations have elapsed
```

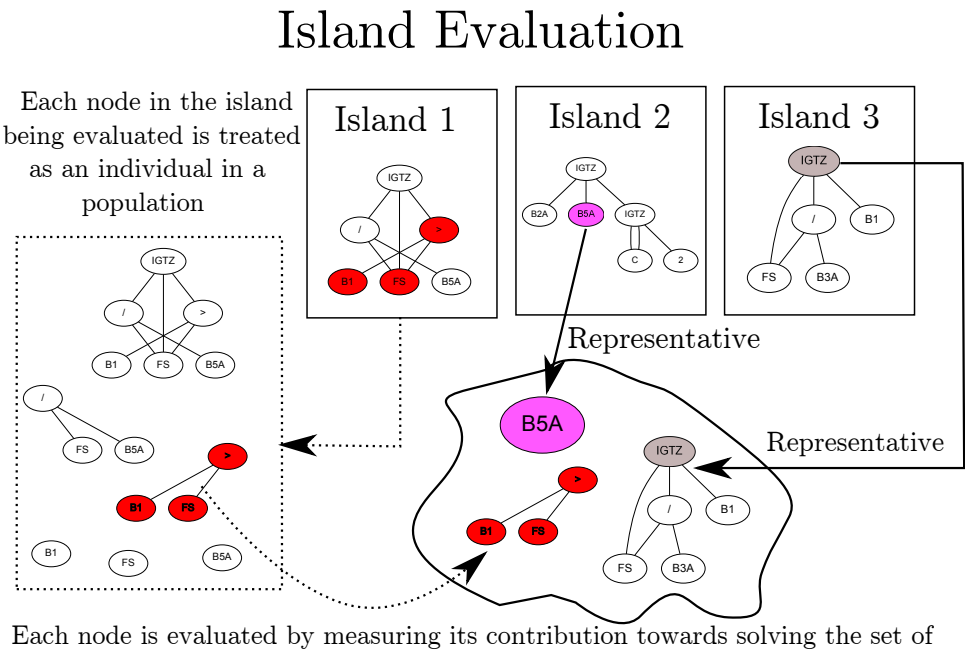


Figure 6.2: Measuring an islands contribution to the ecosystem.

the fitness metric's calculation.

$$fitness_{ij} = \sum_{k=1}^{p} \Delta bins_{ijk} \tag{6.1}$$

where $fitness_{ij}$ is the fitness of $island_i$ $node_j$ evaluated across all training problem instances $k = 1, \ldots, p$

$$\Delta bins_{ijk} = \begin{cases} best_p - bins_{ijk} & \text{:if } bins_{ijk} < best_p \\ 0 & \text{:otherwise} \end{cases} \tag{6.2}$$

where $\Delta bins_{ijk}$ is the difference in the number of bins used to solve problem instance k using island i node j and the best result obtained using the other islands given by $best_p$

$$best_p = \min\left(best_{i_{bins_p}}\right) \ \forall i \in \{1, \ldots, n\} : i \neq x \tag{6.3}$$

where x is the id of the island being evaluated, $best_{i_{bins_p}}$ is the number of bins required to pack problem p using the node with the best fitness from island i and n is the number

of islands in the ecosystem. Each island is evaluated in turn. Figure 6.2 shows how each of the six nodes from island 1 (along with their successor nodes) that constitute the 6 *heuristics* are decomposed. Each node from the island being evaluated is placed into a set of nodes containing one representative from each of the other islands. The node selected as a representative from each of the other islands is that which was awarded the highest fitness score during the previous iteration. All nodes in an island have their fitness value recalculated after the island is mutated as follows.

- Each of the 685 training problem instances are solved using each of the representatives from the other islands.
- If the node being evaluated is able to solve any of the same instances using fewer bins then the improvement in the number of bins is added to its fitness score.
- Only problem instances where an improvement is seen are used for determining a node's fitness.
- Once all nodes in an island have been evaluated the collective ability of the ecosystem is evaluated by measuring the total number of bins required to pack all training instances when the best node from each island is applied greedily to all training instances.
- If the total number of bins required increases from the value obtained during the last iteration then the mutation is undone and the island is reverted to its previous state.

This process then repeats with the other islands. The following section details the experiments conducted and results obtained using the system described above.

6.4 Experiments and Results

The objective of the experiment was to co-evolve a set of cooperative heuristics using half of the problems in Problem Set A for training that when applied greedily to the unseen 685 problem in the test set were able to collectively outperform any of the individual constituent heuristics. The partitioning of the problem instances from Problem

Set A into training and test sets was identical to those used in the previous chapter. The experiment was performed 30 times with each run terminated after 500 iterations. The results presented here only show the first 250-260 generations as no improvement was observed after this point. Each islands SNGP structure was initialised as described in Section 5.4 using 12 randomly selected function nodes. The software was implemented in Java and executed on a high performance cluster comprising of 18 servers each equipped with dual, quad-core cpu's with 16Gb ram running Fedora 12.

The results of the experiment described here, designed to generate a set of cooperative heuristics which when applied greedily to the training problems collectively minimise the number of bins required, are shown in Table 6.1 The number of heuristics evolved is an emergent property of the system and is not predefined. All heuristics contribute towards the combined improvement. The number of optimal solutions found and the number of bins utilised by each new heuristic are shown. None of the individual heuristics evolved by the island model perform as well as the single heuristics generated in the previous chapter. Collectively the system is able to solve 7.9% more problem instances optimally using 38% fewer extra bins than the best single heuristic evolved in the preceding chapter and 30% more optimally than the best human designed heuristic using 64.7% fewer extra bins. Note that an optimal packing of all instances in the test set would cumulatively yield 60257 bins. It is clear that the evolved heuristics outperform the human designed ones in terms of both metrics used. In order to provide a further comparison, the best set of heuristics obtained were applied greedily to the much larger set of 15830 problem instances in Problem Set C available from [111] The best single evolved heuristic used 7.8% fewer extra bins than were required by ADJD which was the best human designed heuristic on these problems. ADJD used 18541 extra bins that the known optimal The set of six cooperative heuristics collectively required 18.7% fewer extra bins than were needed using the best human designed heuristic ADJD and 61% fewer than FFD which required 38650 extra bins.

Figures 6.3 and 6.4 show the performance of the island model during training. The darkest line at the top of Figure 6.3 and the bottom of Figure 6.4 show the results obtained if the heuristics are applied greedily to each instance. The other plots on each

Table 6.1: Results obtained by each of the cooperating heuristics evolved by the island model when applied individually (H1 - H6) and greedily (Combined) to the unseen 685 test problems. The right half of the table repeats the results presented in the previous chapter where a single heuristic (HGEN) was generated. Also shown are the results attained using 4 deterministic human designed heuristics

Evolved Heuristic	Optimal Solutions	Extra Bins
H1	332	692
H2	476	820
H3	465	363
H4	420	500
H5	267	850
H6	471	409
Combined	**559**	**159**

Designed Heuristic	Optimal Solutions	Extra Bins
FFD	393	1088
DJD	356	1216
DJT	**430**	**451**
ADJD	336	679
HGEN	**518**	**257**

graph show the results obtained on the training set by each of the individual constituent heuristics evolved by the system.

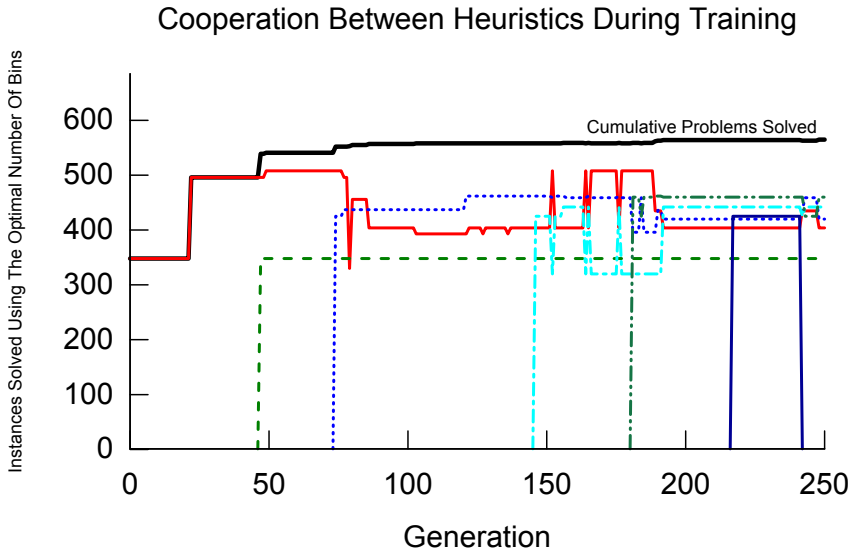


Figure 6.3: Number of problems solved individually and cumulatively from 685 problem instances used during training

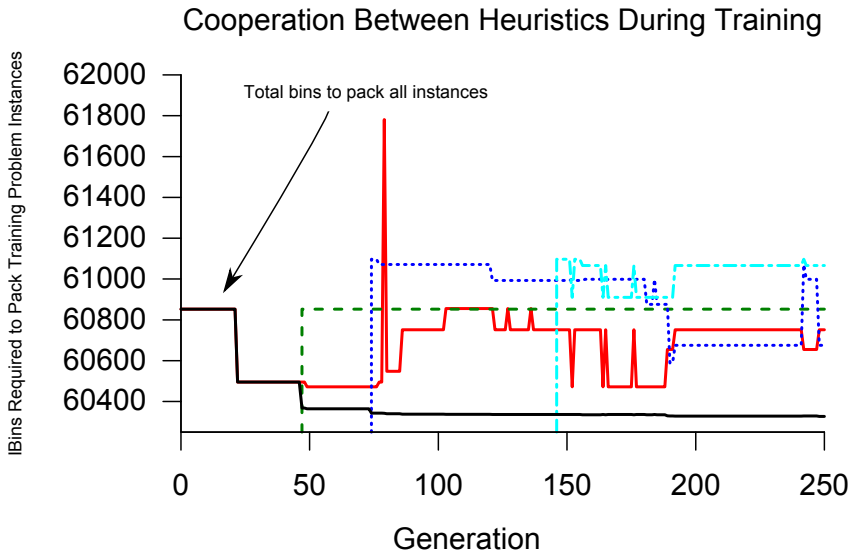


Figure 6.4: Number of bins used individually and collectively during training for all 685 problems in the training set.

Figure 6.5 depicts one of the best sets of 6 heuristics evolved during this experiment which is the set used to obtain the results shown here. Although the results shown are for one individual run, as was the case with the first experiment nearly all of the 30 runs conducted converged to produce identical results which are omitted to increase clarity (27 out of the 30 runs conducted produced sets of heuristics which gave the same results when evaluated using both metrics).

Tables 6.2 and 6.3 show the number of problems that are solved that require the specified number of extra bins than the known optimal for both problem set A and problem set C. In the previous chapter it was stated that although a single heuristic could be generated that outperformed any of the human designed heuristics when evaluated using the number of problems solved or the total number of bins used as a metric that the benchmark heuristics did perform better in some cases. This is not the case here where the combination of heuristics if applied greedily solve 97% of the problem instances in problem set A using no more than 1 extra bin and require at worst 5 extra bins for any individual problem instance.

Table 6.2: Problems solved and extra bins required on problem set A using the evolved collective of heuristics

	Number of Problems Solved Requiring δ extra bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
H1	680	362	165	78	36	22	12	9	1	5	0
H2	938	237	58	34	14	10	18	4	6	1	50
H3	939	303	57	39	15	11	2	2	1	1	0
H4	840	312	95	58	32	17	8	5	1	1	1
H5	551	432	179	100	47	24	12	15	5	5	0
H6	945	264	72	46	18	9	2	8	5	1	0
Combo	1126	202	26	12	2	2	0	0	0	0	0

Table 6.3: Problems solved and extra bins required on problem set C using the evolved collective of heuristics

	Number of Problems Solved Requiring δ extra bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
H1	1749	12101	1435	251	139	125	20	6	3	1	0
H2	1305	9555	2449	689	599	261	108	80	220	110	454
H3	972	10882	1615	762	478	315	178	135	59	30	404
H4	310	8102	2099	1603	932	369	522	533	365	145	850
H5	531	11507	2278	738	526	49	18	34	144	5	0
H6	448	10428	2040	1063	753	122	176	162	200	36	402
combo	3145	10985	1223	318	117	35	7	0	0	0	0

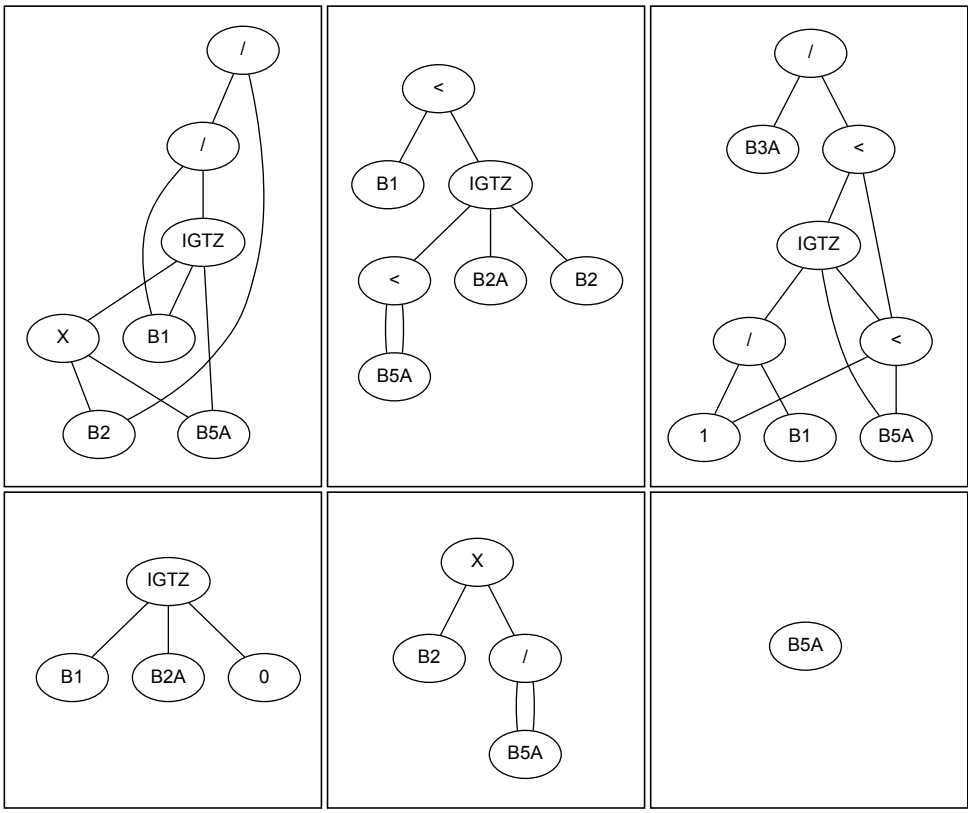


Figure 6.5: An evolved set of cooperative heuristics with unused nodes omitted for clarity. The set shown is an example a set of heuristics that gave the best collective performance when greedily evaluated on the 685 problem instances in the test set

6.5 Conclusions

Using a form of cooperative co-evolution we were able to generate a *set* of novel heuristics that interact to collectively minimise the number of bins used across a large diverse set of problem instances.

The approach is novel in its use of cooperative co-evolution to find sets of reusable heuristics that collectively generalise across the range of problem instances that they were evaluated on. The results highlight the utility of using a HH to combine simple deterministic heuristics in order to exploit their combined strength. The utility of the evolved heuristics on the newly generated problems in problem set C enforce the reusability and generality of the evolved set of heuristics. It was observed during the

study that many of the problem instances tackled are solved easily by multiple heuristics and the fitness metric used to collectively evaluate the set of heuristics disregarded any such instances. The following Chapter introduces a rival approach, loosely based on the Immune System, that aims to increase the efficiency of the co evolution process by limiting the problem instances included to those that are not solved easily by multiple heuristics.

Chapter 7

An Artificial Immune System Inspired Generative Hyper-heuristic

This chapter introduces a generatice hyper-heuristic which uses an Artificial Immune System (AIS) as a replacement for the island model used in the hyper-heuristic presented in the previous chapter. Inspired by an analogy to the immune systems ability to sustain a network of self stimulating antibodies that cover the pathogen space the novel hyper-heuristic introduced in this chapter uses concepts inspired by models of the immune system to sustain a network of complementary heuristics that cover the heuristic search space. The AIS model is shown to have a number of advantages over the island model in terms of efficiency and responsiveness while maintaining solution quality when evaluated on the same problem instances.

7.1 Contribution

The research presented in this chapter is based on work published during the period of study in [110, 112, 116] and was funded in part by EPSRC grant P/J1021628/1 Real World Optimisation with Life-Long Learning. The chapter introduces a hyper-heuristic which uses concepts inspired by models of the natural immune system. The novelty of the approach is in its use of an Artificial Immune System (AIS) as a hyper-heuristic and in the application of concepts taken from Immune Network Theory (INT)

for concurrently solving multiple instances of the BPP. The AIS responds rapidly to problem instances of similar characteristics to which it has been exposed previously yet remains plastic, allowing it to continuously learn and refine its current knowledge when presented with newly introduced problems. The AIS efficiently retains knowledge of the problem / heuristic space that it has been exposed to using a minimal network of interacting heuristics and problem instances that map to the search space. To the best of the authors knowledge, the work presented in this chapter is the first example of an immune-inspired hyper-heuristic optimisation system. The work presented in this chapter spawned a large set of newly generated benchmark problem instances introduced in [112] and described earlier in Chapter 3.

7.2 Motivation

Previous chapters have shown how different heuristics work best on certain classes of problems which occupy different parts of the problem space and that sets of heuristics can be co-evolved to cover that space more effectively than any single heuristic. Inspired by the behaviour of the immune system this chapter introduces an Artificial Immune System (AIS) hyper-heuristic that is used to generate collectives of heuristics which individually operate best on niche areas of the problem space whilst collectively adding to the overall utility of the system.

Advantages of the Immune Model Compared with the Island Model The island model required a training phase that was used to generate sets of heuristics that could then be applied to similar problem instances without modification. This stage was computationally expensive as each island sustained a population of heuristics and each heuristic was required to solve each problem instance that it was being used to solve at each iteration. When evaluating the islands as a collective, each island contributed only its best heuristic which ultimately meant that the majority of the heuristics sustained in each island were redundant once the learning stage had terminated. In contrast the AIS evolves populations of heuristics in a single connected model that only allows heuristics to persist if they add to the collective performance of the system. Further-

more a novel affinity metric minimises the requirement to solve all problem instances that the system is being evaluated on which substantially increases the efficiency and scalability of the system.

Once trained, the heuristics sustained by the island model were used to solve a test set of problem instances that had similar characteristics to those problem instances that it had encountered during training. The system was unable to adapt to new problem instances that differed in characteristics to to those that it had already encountered. In contrast the AIS is shown to be highly responsive when faced with new problem instances and highly adaptive when supplied with continually changing sets of problems. The AIS efficiently maintains a memory of previously encountered problems allowing for an immediate response if problems of similar characteristics are reintroduced.

7.3 Background

The system presented in this chapter, originally described in [116] and improved in [112, 114], is inspired by behaviour exhibited by the natural immune system. The immune system has the following properties that are analogous with those identified as necessary for a search mechanism striving to provide high quality solutions quickly for a range of problems with different characteristics.

- It can adapt its knowledge via evolutionary mechanisms allowing for a more rapid response to newly presented pathogens.
- It efficiently encapsulates the pathogen space using the minimal repertoire of antibodies.
- It exhibits memory enabling it to respond rapidly to previously encountered pathogens.

Artificial Immune Systems (AIS) algorithms have been applied in many domains, including optimisation, robot control and pattern recognition[34, 73]. Unlike other biologically inspired paradigms there is no de-facto model used by AIS practitioners. Of the models used the one most frequently applied to CO problems is *clonal selection*

theory [23] which takes inspiration from the classical immune model, called the *self recognition view*, where antibodies bind only to antigen. Binding results in a cloning process and mutation process, resulting in a proliferation of antibodies around promising search locations for new antibodies that better match the invading pathogen.

As a search and recognition mechanism the human immune system is thought to be capable of detecting any shaped *antigen* using a repertoire of 10^{12} lymphocytes that each produce around $10^5 - 10^7$ identical antibodies. An antibody marks a pathogen for destruction through a process of binding in which a region on antibody known as the *paratope* physically binds with an area on the antigen called the *epitope*. A lock and key analogy is frequently used to describe the binding process in which the strength of the binding (*affinity*) is proportional to the degree of complementarity between the two shapes.

The self recognition model was challenged when Jerne[67] proposed his *self assertive view* of the immune system encapsulated by Idiotypic Network Theory (INT). Jerne suggested that antibodies can also recognise other antibodies, creating a network of stimulatory and suppressive signals which can be sustained in the absence of antigen. The theory, although now largely disputed, explained both the immune response and the existence of immune memory. The system presented in this chapter simultaneously evolves sets of heuristics that are sustained in a network whose dynamics are inspired by the mechanics of INT wherein a novel affinity metric is used to stimulate heuristics and problem instances that efficiently and effectively cover the search space and suppress those that do not add to the utility of the collective.

7.4 An Artificial Immune System Hyper-heuristic

The following section describes the conceptual view of the system before describing the implementation of different elements in detail.

7.4.1 Concept

The hyper-heuristic comprises of three main parts as illustrated by Figure 7.1.

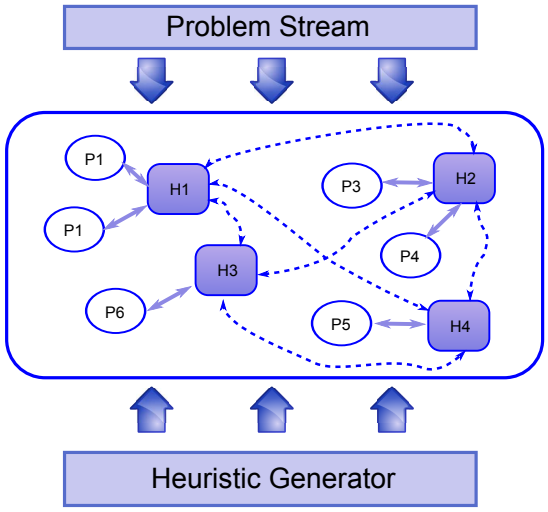


Figure 7.1: A conceptual view of the system: problems are continuously added/removed from the system. The generator continuously injects new heuristics. The dynamics of the system result in a self-sustaining network of heuristics and problems that efficiently cover the search space. Solid lines show *direct* interactions, dashed lines represent *indirect* interactions.

- A stream of problem instances.
- A stream of novel heuristics.
- An Artificial Immune System inspired by Idiotypic Network Theory.

The system is designed to run continuously with problem instances and heuristics continually added to the network. The AIS itself consists of a network composed of interconnected problems and heuristics that interact with each other based on a novel affinity metric. The system is tested on both static sets of problem instances and continually changing sets where the set of problem instances presented to the system is continually changed over time. Throughout the remainder of this chapter the following terminology is used to describe the different elements of the system.

- $\mathcal{U}$ - the theoretical set of all possible problems in the BPP domain.
- $\mathcal{U}'$ - a subset of $\mathcal{U}$ that contains a specific set of problems, e.g problem set A or B

- $\mathcal{E}$ - the current environment, i.e. the set of problems we are currently interested in solving, i.e. $\mathcal{E} \subset \mathcal{U}' \subset \mathcal{U}$
- $\mathcal{E}^*$ - the set of *all* problems to which the system has been exposed during its lifetime
- $\mathcal{N}$ - the immune network, comprised of a set of problems and a set of heuristics
- $\mathcal{P}$ - the set of *problems* currently sustained in the immune network $\mathcal{N}$, i.e. $\mathcal{P} \subset \mathcal{E}^*$
- $\mathcal{H}$ - the set of *heuristics* currently sustained in the immune network $\mathcal{N}$

The immune network sustains a minimal repertoire of heuristics and a minimal repertoire of problems that provide a representative map of the problem space to which the system has been exposed over its lifetime. From a problem perspective, the network does not retain all problem instances from the problem stream but a representative set that is sufficient to map the problem space. From a heuristic perspective, only heuristics that provide a unique contribution in that they produce a better result on at least one problem than any other heuristic are retained. This is represented conceptually in figure 7.2. In this diagram, the first figure (a) shows a set of problems $\mathcal{E}$ that the system is currently exposed to. (b) shows a set of heuristics $\mathcal{H}$ that collectively cover the problems in $\mathcal{E}$. The problems shown in red are solved equally by two or more heuristics. H2 is subsumed in that it cannot solve *any* problem better than another heuristic. In (c), H2 is removed as it does not have a niche in solving problems; problems P1 and P2 are removed as the do not have a niche in describing the problem space[1]. A competitive exclusion effect is observed between heuristics (and also between problems) that results in efficient coverage of the problem space.

The AIS is continuously supplied with novel heuristics that are generated using the initialisation procedure of Single Node Genetic Programming, described over the previous 2 chapters. Rather than using genetic operators to evolve heuristics the AIS controls the dynamics of the network sustaining only those heuristics that implicitly stimulate each other. Unlike the Island Model presented in the previous chapter the

[1] Although these problems have been removed from the network, they can still be solved by the system as heuristics H1 and H3 remain in the network

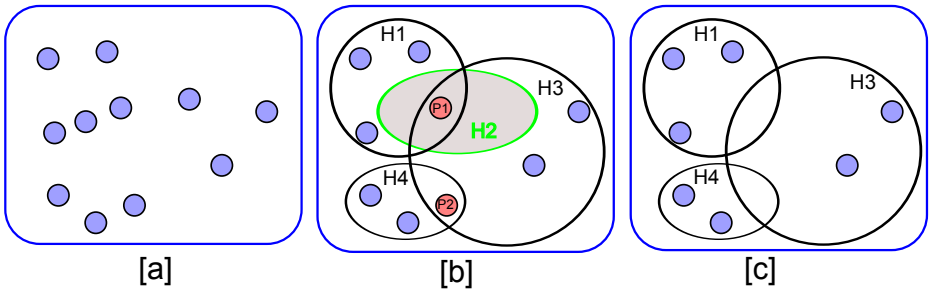


Figure 7.2: The left-hand figure shows the problems that the system is currently exposed to $\mathcal{E}$. The middle diagram shows a set of generated heuristics that cover the problems in $\mathcal{E}$. The problems shown in red are equally solved by one or more heuristics and therefore not required to map the problem space. The heuristic in green is redundant as it does not have a niche. The right-hand figure shows the resulting *network* $\mathcal{N}$ that sustains the minimal set of problems and heuristics required to describe the space.

AIS sustains not only a minimal repertoire of heuristics but also a minimal set of problem instances that encapsulate the larger problem space. The minimal repertoire of network components sustained by the AIS is shown to have statistically equivalent performance, in terms of solution quality, when applied to the same set of benchmark problem instances as the island model. However the AIS has the following advantages.

- It is highly responsive when initially presented with a new set of problem instances.
- It responds rapidly to change by incorporating new heuristics and problem instances as required and removing those that do not add to the collective utility of the system.
- It exhibits memory of problems previously encountered allowing for a rapid response when presented with similar problem instances.
- It is more scalable due to the method of efficiently encapsulating the problem space.

7.4.2 Implementation

In this section the key components of the system, outlined in the Pseudo-code given in Algorithm 5, are described.

Algorithm 5 AIS-HH Pseudo Code

Require: $\mathcal{H} = \emptyset$:The set of heuristics

Require: $\mathcal{P} = \emptyset$:The set of current problems

Require: $\mathcal{E} =$ The set of problems to be solved at time t

1: **repeat**
2: *optionally* replace $\mathcal{E}$: $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup \mathcal{E}$
3: Add n_h randomly generated heuristics to $\mathcal{H}$ with concentration c_{init}
4: Add n_p randomly selected problem instances from $\mathcal{E}$ to $\mathcal{P}$ with concentration c_{init}
5: calculate $h_{stim} \forall h \in \mathcal{H}$ using Equation 7.1
6: calculate $p_{stim} \forall p \in \mathcal{P}$ using Equation 7.2
7: increment all concentrations (both $\mathcal{H}$ and $\mathcal{P}$) that have concentration $< c_{max}$ and stimulation > 0 by Δ_c
8: decrement all concentrations (both $\mathcal{H}$ and $\mathcal{P}$) with stimulation ≤ 0 by Δ_c
9: Remove heuristics and problems with $concentration \leq 0$
10: **until** $stopping\ criteria\ met$

The Artificial Immune System

The AIS component is responsible for constructing a network of interacting heuristics and problems, and for governing the dynamic processes that enable heuristics to be incorporated or rejected from the current network.

The network $\mathcal{N}$ sustains a set of interacting heuristics and problems. Problems are directly stimulated by heuristics, and vice versa. Heuristics are indirectly stimulated by other heuristics through a competitive exclusion effect where different heuristics compete for the stimulation provided by a limited number of problem instances. Each heuristic h can be stimulated by one or more problems and the total stimulation received by a heuristic is the sum of its affinity with each problem currently in the net-

work ($\mathcal{P}$). A heuristic h has a non-zero affinity with a problem $p \in \mathcal{P}$ if and only if it provides a solution that uses fewer bins than all of the other heuristics currently in $\mathcal{H}$. The affinity between $p \leftrightarrow h$ is equal to the improvement in the number of bins used by h compared to the next-best heuristic. In all other cases (ie where more than one heuristic provides the *best* result or where the heuristic provides a worse solution than another heuristic) the heuristic receives no stimulation. This is expressed mathematically by equation 7.1, in which $\mathcal{H}'$ is the set of heuristics currently in the system, *excluding* the heuristic h currently under consideration, i.e. $\mathcal{H}' = \mathcal{H} - h$.

$$h_{stim} = \sum_{p \in \mathcal{P}} \delta_{bins} \begin{cases} \delta_{bins} = \min\left(bins_{\mathcal{H}'_p}\right) - bins_{h_p} & : \text{ if } \min\left(bins_{\mathcal{H}'_p}\right) - bins_{h_p} > 0 \\ \delta_{bins} = 0 & : \text{ otherwise} \end{cases} \tag{7.1}$$

As the affinity between a problem and a heuristic is symmetrical, then the stimulation of a problem is simply the affinity between the problem and the heuristic that best solves it. A problem for which the best solution is provided by more than one heuristic receives zero stimulation. Thus, unlike heuristics, a problem can only be stimulated by one heuristic. This is expressed mathematically in equation 7.2. Note that in the sum expressed in this equation, at most one term will be non-zero. This causes problem instances for which the *best solution found so far* is easily attained by more than one heuristic to be removed from the system although at least one of the heuristics that provided that solution will be sustained.

$$p_{stim} = \sum_{h \in \mathcal{H}} \delta_{bins} \begin{cases} \delta_{bins} = \min\left(bins_{\mathcal{H}'_p)}\right) - bins_{h_p} & : \text{ if } \min\left(bins_{\mathcal{H}'_p}\right) - bins_{h_p} > 0 \\ \delta_{bins} = 0 & : \text{ otherwise} \end{cases} \tag{7.2}$$

The Problem Stream

A stream of problem instances are continually injected into the system. At each iteration n_p problem instances are randomly selected from $\mathcal{E} - \mathcal{P}$ and entered into the AIS with an initial concentration of c_{init}. At each iteration if each newly entered problem receives no stimulation (described by step 6 of Algorithm 5) then its concentration is

reduced by Δ_c. Problem instances that best describe the environment from the perspective of any of the heuristics are stimulated and incorporated into the network. Problem instances for which the best found solution is easily obtained are removed once their concentration's reach 0.

The Heuristic Stream

A stream of newly generated novel heuristics are continually supplied to the AIS. At each iteration n_h heuristics are randomly initialised using the initialisation procedure from SNGP and incorporated into $\mathcal{N}$ with a concentration level of c_{init}. At each iteration if the heuristic(s) receive no stimulation (dont solve any problems better than any other heuristic, described by Equation 7.1) then their concentration is reduced by Δ_c. Heuristics that find niche problems on which they provide better solutions than any of the other heuristics currently in the system are stimulated and incorporated into the network. Heuristics that perform better than heuristics already in the system will indirectly suppress those heuristics causing them to (potentially[1]) be removed from the system. Heuristics that receive no stimulation for $\frac{c_{init}}{\Delta_c}$ iterations are removed.

The method of generating heuristics used is simplified over than used in the previous two chapters. Only the initialisation procedure of SNGP is utilised after which one node is selected at random and the associated sub-tree structure is retained as a single fixed heuristic that undergoes no further mutation. No evolutionary operators are used to improve upon the randomly initialised heuristics. The justification for this is that the heuristic generator's role is simply to provide a continuous source of novel material for potential integration into the network. The dynamics of the network will cause poor heuristics to be eradicated. Also after extensive experimentation conducted with the island model it was observed that often heuristics of reasonable quality were generated randomly which is to be expected given the relatively small number of terminal and function nodes available. This is shown in the results presented in the box plots in Section 5.6 where in at least one run out thirty the best heuristic was obtained during the initialisation stage. The set of function nodes and terminal nodes used to

[1]New problems may be injected during the heuristics lifespan that cause the heuristic to be stimulated.

create new heuristics remains identical to those described over the previous 2 chapters. Randomly generated deterministic constructive heuristics are created using the initialisation process from SNGP which is repeated below for clarity in Algorithm 6.

Algorithm 6 Heuristic Generation

1: Each of the terminal nodes $T \in \{t_1, \ldots t_r\}$ are added exactly once. The terminal nodes are given an integer identification number ranging from $1 \ldots r$.
2: A number, n, of function nodes are selected at random from the set of all function nodes $F \in \{f_1, \ldots, f_s\}$ and given an identification number ranging from $r + 1$, $\ldots$ to $r + n$. This allows for the possibility of duplicate function nodes within the population or for SNGP structures with function nodes omitted.
3: The function nodes have all their child nodes assigned at random from nodes with a lower id thus preventing any infinite looping.
4: A single node is chosen at random to be the root node for the heuristic which undergoes no further modification.

Deviation from the INT Model

The affinity metric implemented does not replicate the original AIS model described by Farmer et al. [46]. The reason for this is one of efficiency. In the original model there is an explicit calculation of affinity that is calculated by summing the affinity exerted on an antibody (heuristic) by all of the other antibodies in the system. Preliminary investigations showed that this quickly lead to a dramatic increase in the number of heuristics sustained and exerted no explicit pressure on the collective ability of the system to improve. If the dynamics of this model were implemented then a heuristic would be stimulated by all of the other heuristics in the system if it performs better than each heuristic on at least some of the problem instances in the environment. The heuristic may however provide no benefit to the collective of heuristics sustained. This undesirable effect is illustrated in Figure 7.2 where, for example, heuristic H2 is redundant to the collective utility of the system. In this scenario H2 would remain in the network as it does have an affinity with each of the other 3 heuristics, solving some of the problem instances better than each if considered in isolation. However H2, if con-

sidered along with the other heuristics as a collective, provides no overall improvement to the system and is simply a drain on resources.

7.5 Experiments and Results

The remainder of this chapter uses concepts, results and conclusions that were obtained from applying both systems, the original AIS [116] and the improved system [112]. In the later of these publications [112] results presented indicate that while both systems have identical utility in terms of solution quality the later achieves these results using a more efficient and scalable approach. The system is evaluated on 2 large test suites of BPP comprising of 5338 problem instances; Problem Set A, taken from the literature and Problem Set B newly generated for [112]. These problem sets are described previously in Section 3.3.7 and Section 3.3.6 respectively.

- The utility of the system compared to similar hyper-heuristic approaches.
- The adaptability and responsiveness of the network in terms of its ability to quickly adapt when presented with new unseen problem instances.
- The ability to retain memory of previously encountered problem instances.
- The efficiency and scalability of the system in maintaining knowledge using a minimal repertoire of network components.

Experiments were conducted using the model described by Algorithm 5 using data drawn from the two sets of data described in section 3.3, problem sets A and B. Unless specifically stated the default parameters used for all experiments were as shown in Table 7.1. These parameters were set following an initial period of empirical investigation.

7.5.1 Utility of System in Comparison to the Island Model

The AIS is benchmarked against the same set of problems used to evaluate the Island model introduced in the previous chapter to obtain an indication of the *quality* of results

Parameter	Description	Value
n_p	number of problems added each iteration from $\mathcal{E}$	30
n_h	number of new heuristics added each iteration	1
c_{init}	initial concentration of added heuristics/problems	200
Δ_c	variation in concentration based on stimulation level	50
c_{max}	maximum concentration level	1000

Table 7.1: Default parameter settings for the AIS-HH

it provides. Comparisons to the benchmark human designed deterministic heuristics used in previous chapters are also given.

Problem set A

The island model presented in the previous chapter involved a training phase, in which the algorithm was trained on a set of problems and performance evaluated on a separate test set. Although the AIS does not have a training phase, for consistency and in order to directly compare results, the same procedure is adopted:

- Problem set A (1370 problems) is split into two equal sized sets (adding every second problem to the test set)
- The AIS is executed for 500 iterations using the training set as the environment $\mathcal{E}$
- The system is stopped and the heuristics sustained at the end of the training phase are applied to the unseen problems from the test set (685) and the number of problems solved and bins utilised recorded.

Table 7.2a directly compares the result obtained by the AIS to the Island model introduced in the previous chapter. A further experiment was run using the AIS where $\mathcal{E}$ was set to the full set of 1370 problems in A rather than a reduced set of 685 problems. To obtain a comparison to the Island model each algorithm was run using the complete set of 1370 problems with no training stage. These results are given in table

Table 7.2: A comparison of results obtained on a static dataset of 685 problems taken from A using a) single heuristics and b) collaborative methods.

(a)

Single Deterministic Heuristics		
Heuristic	Problems Solved	Extra Bins
FFD	393	1088
DJD	356	1216
DJT	**430**	**451**
ADJD	336	679

(b)

	Collaborative Heuristic Models							
	Problems Solved				Extra Bins			
	min	max	mean	sd	min	max	mean	sd
Island	552	**559**	557	1.4	**159**	164	162	1.4
AIS	559	**559**	559	0	**159**	159	559	0

7.3 . The results confirm that the two systems produce solutions of identical quality on a static data-set. The remainder of this chapter illustrates a number of advantages that the AIS exhibits when compared to the previous approach. Specifically, the system is shown to be scalable; it significantly reduces computation time compared to previous approaches; it is shown to adapt efficiently to unseen problems and rapidly changing environments (sets of problem instances).

Further analysis is given in table 7.4 which shows the number of problem instances solved using the specified number of bins more than the known optimum for the complete set of 1370 problems in A. The AIS clearly outperforms the individual human designed deterministic heuristics — many of these perform particularly poorly on certain problem instances. On the other hand, the evolved set of cooperative heuristics retained by the AIS solves 97% of problem instances using no more than 1 extra bin.

Problem Set B

The experimental procedure defined above was repeated using the new and larger problem set B in order to ascertain the systems performance on this new set of problems and to provide a baseline for further experimentation.

The system was executed 30 times with each run conducted over 100,000 iterations using the full set of problems as the environment $\mathcal{E}$ and the default parameters as specified in table 7.1. A summary of the results is given in Table 7.5 which also contrasts the

Table 7.3: A comparison of results obtained on the complete set of 1370 problems in A using a) single heuristics and b) collaborative methods. The results presented also demonstrate the efficiency of the AIS in sustaining the network using both a minimal repertoire of heuristics and problem instances.

(a)

Single Deterministic Heuristics		
Heuristic	Problems Solved	Extra Bins
FFD	788	2142
DJD	716	2409
DJT	**863**	**881**
ADJD	686	1352

(b)

	Collaborative Heuristic Models							
	Problems Solved				Extra Bins			
	min	max	mean	sd	min	max	mean	sd
Island Model	1120	**1126**	1125	1.1	**308**	316	308	1.4
AIS	1125	**1126**	1126	0.3	**308**	309	308	0.3

Heuristics Retained				Problems Retained			
min	max	Mean	SD	min	max	Mean	SD
6	8	7.1	0.7	26	57	36.9	6.4

Table 7.4: Extra bins (δ) required by the AIS compared to 4 deterministic heuristics on the complete set of 1370 benchmark problem instances

	Number of Problems Solved Requiring δ Extra Bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
FFD	788	267	78	83	39	16	18	9	18	4	50
DJD	716	281	119	58	48	36	10	16	23	3	60
DJT	863	331	90	26	30	15	11	2	1	1	0
ADJD	686	368	153	76	38	22	12	9	1	5	0
AIS	1126	202	26	12	2	2	0	0	0	0	0

Table 7.5: Comparison of the number of bins required by the AIS and the benchmark heuristics on problem set C

Heuristic	Total Bins	Extra Bins Than Optimal	Problems Solved Optimally
Optimal	320445	0	3968
FFD	327563	7118	491
DJD	330447	10002	920
DJT	325743	5298	1158
ADJD	323566	3121	1279
AIS	**322820**	**2375**	**1983**

results against those achieved using 4 human designed deterministic heuristics. These results are analysed further in Table 7.6 which gives the number of problems solved using the specified number of bins greater than the known optimal by each of four deterministic heuristics and the AIS.

Table 7.6 also demonstrates the relative complexity of the problem instances in B when contrasted to the standard benchmarks in A, with respect to the standard set of deterministic heuristics. For example, on problem set A, FFD was shown to solve 56% of the 1370 problem instances using the known optimal number of bins. In contrast, on problem set B, it only manages to solve 12% optimally. The AIS solves 82% of the problems in A optimally, compared to only 50% of the problem instances in B.

Note that the final evaluation of each of the 30 runs gave exactly the same result in terms of the number of bins required to pack each of the problems in B (although the heuristics and problems sustained in each run differed). One of the runs was selected at random and the results obtained by the final set of heuristics for each instance in B were retained for use in the remaining experiments as a benchmark for the problem set[1].

[1] In the graphs and tables shown in the remainder of this chapter the term *best* refers to this benchmark value. This allows plots to be displayed on an equal scale where the number of problem instances and the *optimal* number of bins dramatically varies from iteration to iteration.

Table 7.6: Extra bins (δ) required by the AIS-HH compared to 4 deterministic heuristic on the new set of 3968 problem instances in Problem Set C

	Number of Problems Solved Requiring δ extra bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
FFD	491	2364	442	208	196	51	22	34	68	19	73
DJD	920	1552	468	248	191	100	92	66	57	34	240
DJT	1158	1936	414	141	85	76	52	35	9	2	60
ADJD	1279	2398	209	38	33	8	2	1	0	0	0
AIS-HH	1983	1708	201	44	27	5	0	0	0	0	0

7.5.2 Parameter Tuning

A brief investigation of the impact of three of the main system parameters is conducted to determine their influence and justify the default settings.

Concentration c_{init}

The effect of varying the initial concentration of problems and heuristics is illustrated in Figure 7.3 which shows the results obtained when the AIS-HH was run 30 times for each of $c_{init} \in \{50, 100, 200, 500, 1000\}$. The system was halted after 100,000 iterations. Each box plot summarises the 30 runs conducted. The vertical axis shows the number of bins more than the *best* result that the AIS-HH achieved on problem set B as described previously and presented in Tables 7.5 and 7.6. For $c_{init} < c_{max}/2$, increasing the initial concentration improves performance — the increased *initial* concentration increases the time period that both heuristics and problem instances can be sustained without stimulation, thus increasing the probability of eventually finding a heuristic-problem pairing that is mutually stimulatory. However, as $c_{init} \rightarrow c_{max}$, the effect is reversed; newly introduced heuristics dominate due to their larger concentration, potentially suppressing previously established heuristics.

Number of problems added per iteration n_p

The parameter n_p describing the number of problems presented to the system each iteration is key in that it has significant impact on the number of calculations that

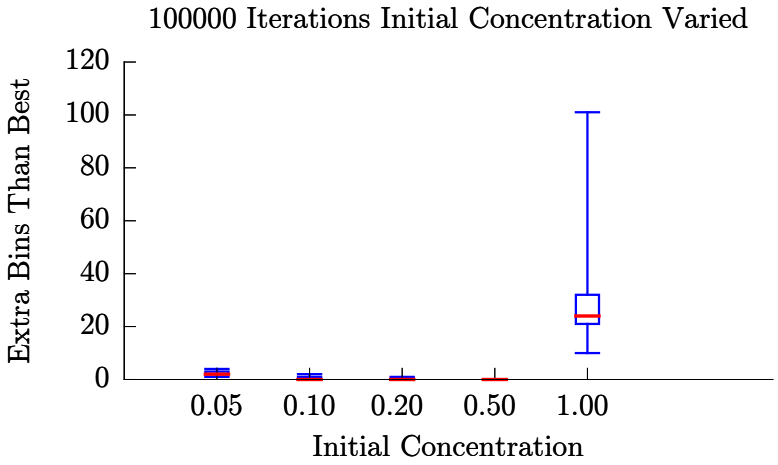


Figure 7.3: The effect of varying the initial concentration c_{init}. The concentration c_{init} on the x-axis is plotted as fraction of c_{max}

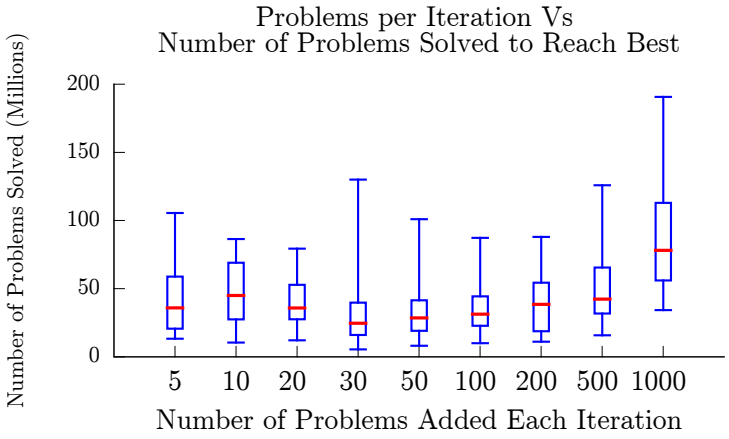
.

need to be made at each iteration of the algorithm. Each iteration, the number of new calculations C that needs to be performed is given by:

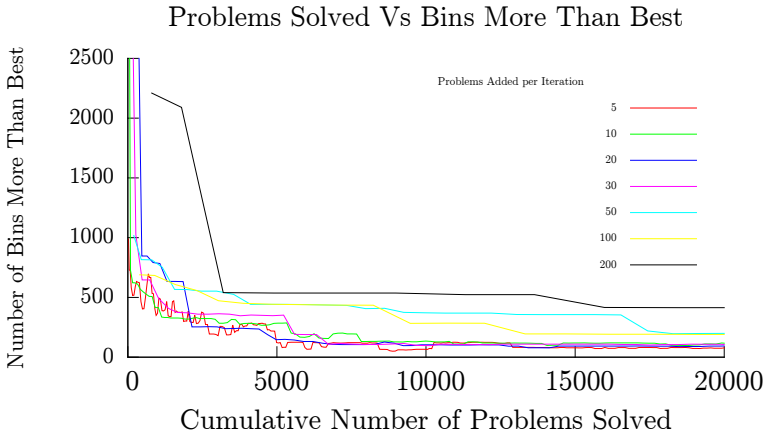
$$C = (n_p \times |\mathcal{H}|) + (n_h \times |\mathcal{P}|) \tag{7.3}$$

The first term is required to determine the result of applying all heuristics in the system to the new problems just introduced. The second term determines the results of any new heuristics introduced this iteration on all problems currently in the system.

To understand the influence of n_p, the model was executed 30 times for each of 6 different values of $n_p \in \{30, 50, 100, 200, 500, 1000\}$. Each iteration, the cumulative number of calculations undertaken is recorded. The model was allowed to run until the results obtained on $\mathcal{E}$ converged to the *best* result known for the system on problem set B. Figure 7.4a summarises the results obtained over 30 runs for each parameter setting. The figure shows that increasing n_p. i.e. the number of problem instances presented each iteration has an adverse affect, increasing the overall number of calculations required to achieve the same result. The default value of 30 appears a reasonable choice. Figure 7.4b shows a single run of the algorithm truncated to 20000 calculations.



(a)



(b)

Figure 7.4: Figure 7.4a shows the number of problem instances added per iteration Vs the number of problems solved by the system in order to reach the best solution. Each box plot shows the results obtained over 30 runs. Figure 7.4b depicts the total number of problems solved to reach the best achieved result during the course of a typical run

Number of heuristics added per iteration n_p

Figure 7.5 shows the affect that varying n_h has on the system. For each plot the system was executed for 50000 generations with $\mathcal{E} = B$ using default parameter settings with the exception of n_h which was fixed for the duration of each plot as shown.

When adding a single heuristic each iteration, a smooth increase in performance is observed over time, and the system converges to the best known result, despite a slow start. Adding a larger number of heuristics per iteration improves the initial performance due to an increased probability of finding good solutions. However over a longer time scale, performance is hindered, causing undesirable fluctuation in the collective capability of the network. In the worst case, when $n_h = 20$, the system fails to converge to the best result.

As n_h increases, the ability of individual heuristics to find niche areas of the problem space becomes more difficult due to increased competition; newly introduced heuristics are unlikely to gain any stimulation due to the decreased probability of the heuristic solving a problem better than any other heuristic resulting in very short lifetimes for each heuristic and thus more unstable behaviour in the system. From a computational perspective, increasing both n_p and n_h also significantly increases the number of calculations required each iteration. This further justifies the choice of $n_h = 1$ as the default value.

7.5.3 Efficiency and Scalability

Any real world hyper-heuristic optimisation system should be both efficient in terms of time taken to achieve a result of acceptable quality, and scalable in terms of the number of problems it can deal with. To determine the scalability of the AIS with respect to $|\mathcal{E}|$ an experiment was conducted in which $|\mathcal{E}|$ was varied, i.e. $|\mathcal{E}| \in \{100, 200, 500, 1000, 2000, 3968\}$. In each case, the problems in $\mathcal{E}$ were randomly selected from problem set B, i.e. $\mathcal{U}' = B$. All other parameters were set to the default values, and the system was run for 50000 iterations over 30 runs.

Table 7.7 shows the mean number of problems and heuristics retained following 50000 iterations of the system. The table also shows the ratio $|\mathcal{P}|/|\mathcal{E}|$, i.e. the fraction

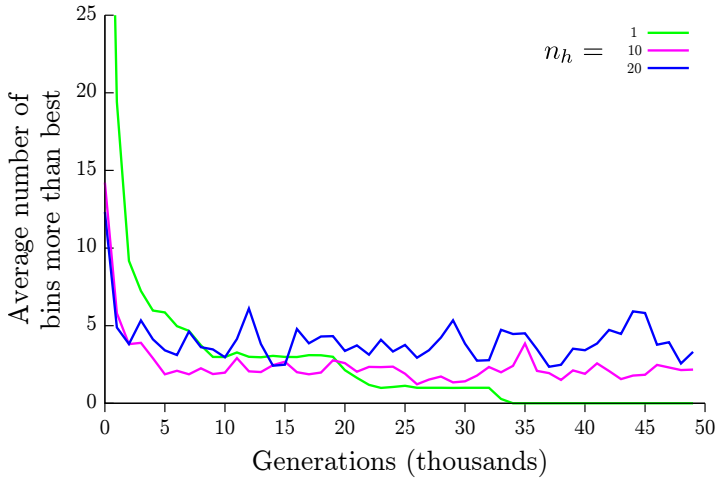


Figure 7.5: Effect of varying the number of heuristics added each iteration (n_h). Results shown are averaged on the problems currently in $\mathcal{E}$ every 1000 iterations

of the problems in the environment retained in the network, and the ratio $|\mathcal{H}|/|\mathcal{P}|$ as the size of $\mathcal{E}$ increases, to indicate how the system scales.

As expected, as $|\mathcal{E}|$ increases , the number of retained problems and heuristics increases. Note however that the fraction of problems retained in relation to the environment *decreases*. The problems in the environment $\mathcal{E}$ represent a sample of problems from the larger $\mathcal{U}' = B$. As $|\mathcal{E}|$ increases, more of $\mathcal{U}'$ is sampled, and thus the system is better able to learn a general representation of $\mathcal{U}'$, hence decreasing the ratio of problems $|\mathcal{P}|/|\mathcal{E}|$ required to represent it. This is also reflected in the sub-linear increase in the number of heuristics required as $|\mathcal{E}|$ increases, again confirming the ability of the system to find heuristics that generalise over the environment.

Table 7.7: The table shows the number of heuristics and problems retained in the network as the size of the environment $\mathcal{E}$ increases. All figures obtained over 30 runs and 50000 iterations

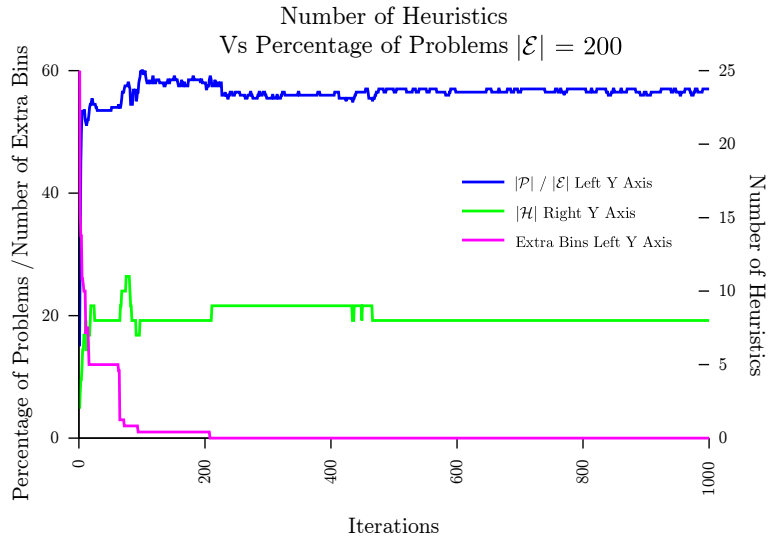
	$\lvert\mathcal{E}\rvert = 100$	$\lvert\mathcal{E}\rvert = 200$	$\lvert\mathcal{E}\rvert = 500$	$\lvert\mathcal{E}\rvert = 1000$	$\lvert\mathcal{E}\rvert = 2000$	$\lvert\mathcal{E}\rvert = 3968$
Mean heuristics retained $\mathcal{H}$	5.40	6.87	9.90	12.40	16.83	21.57
Mean problems retained $\mathcal{P}$	18.73	23.3	33.45	41.5	47.4	59.52
Ratio $\mathcal{P}/\mathcal{E}$	18.73	11.65	6.69	4.15	2.37	1.50
Ratio $\mathcal{P}/\mathcal{H}$	0.29	0.29	0.30	0.30	0.35	0.36

The ratio $|\mathcal{H}|/|\mathcal{P}|$ remains almost constant, indicating the scalability of the system. Figure 7.6 shows a typical run from an experiment for both $|\mathcal{E}| = 200$ and $|\mathcal{E}| = 3968$[1].

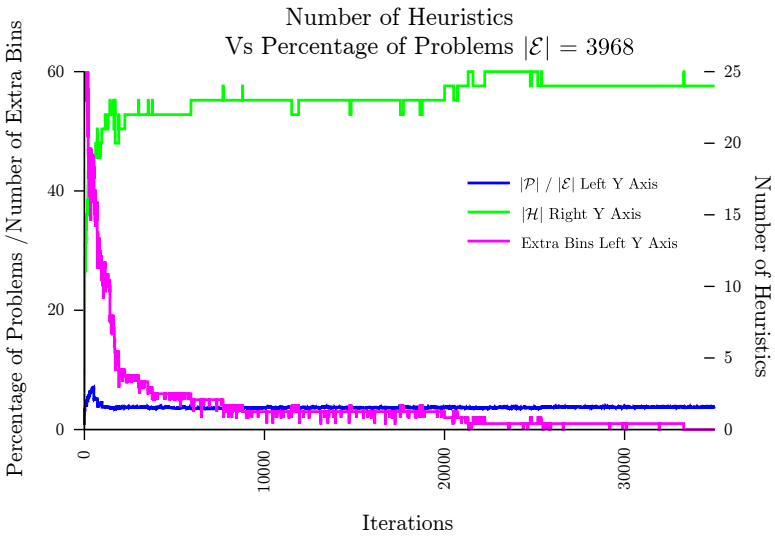
7.5.4 Memory Capabilities of the AIS

In order to demonstrate that the AIS maintains a memory of previously encountered instances using a minimal repertoire of heuristics and problem instances it was tested on dynamically changing problem environments. Note that the term dynamic here refers to the continually changing set of problem instance presented to the AIS and not the more typical definition where the fitness function undergoes change during the course of a run. Two scenarios were investigated, described over the following two sections. The first evaluates the system in an environment where the set of problems presented to the system change every 1000 iterations but where those problems are similar to ones encountered during each of the previous epoch. The second experiment shows how the system responds when faced with an alternating environment which every 500 iterations is switched between two distinctly different sets of problems. This second experiment investigates the ability of the system to retain knowledge of problem instances that were encountered during an epoch separated from the current iteration by

[1] As both heuristics and problems are continually added with sufficient concentration to allow them to survive for at least 3 iterations, then at any iteration, there will be potentially be at most 3 heuristics and 90 problem instances that give no added benefit to the system. The table shows only $\mathcal{H}$ and $\mathcal{P}$ after the run finishes where any unstimulated problems and heuristics are removed thus the discrepancy between the mean for $\mathcal{E} = 200$ being 11.65% in the table and 60% in the figure



(a)



(b)

Figure 7.6: A comparison of the performance of the system applied to different sized data sets. Figure 7.6a plots a typical run for $|\mathcal{E}| = 200$. Figure 7.6b plots a typical run for $|\mathcal{E}| = 3968$Each graph shows the percentage of problems solved, the total number of extra bins used and the total number of heuristics sustained by the network plotted against the number of iterations)

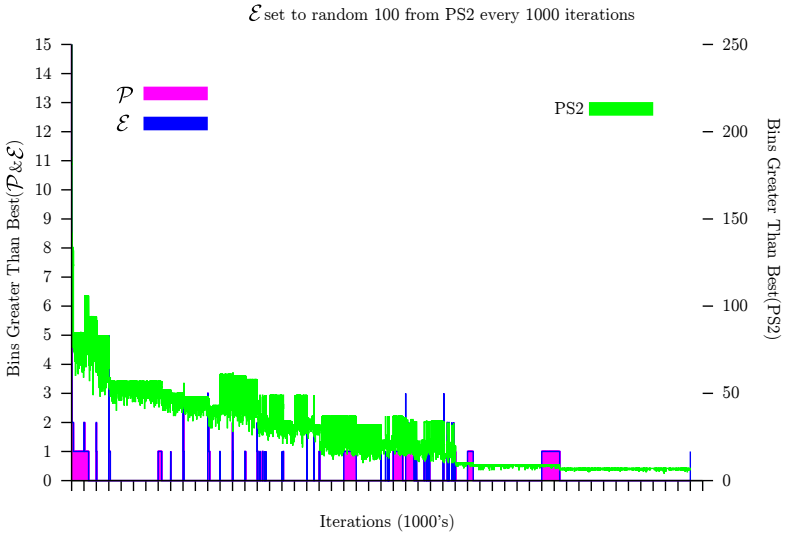
a significant duration whilst continuing to improve over the complete set of problem instances presented.

Memory and Learning: Response to new problems from a similar dataset

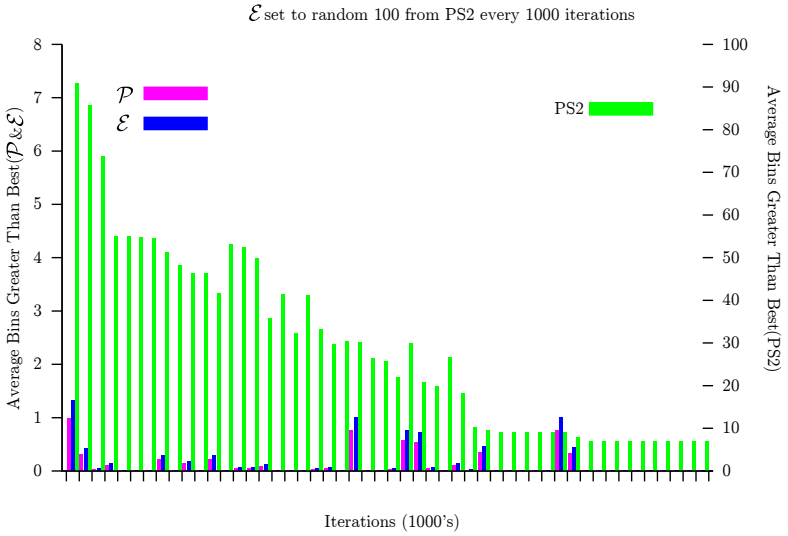
Consider the case in which $\mathcal{U}' = B$, i.e. the set of 3968 novel problem instances. At $t = 0$, $\mathcal{E}$ consists of a set of $|\mathcal{E}|$ problems drawn randomly from $\mathcal{U}'$. Every 1000 iterations, $\mathcal{E}$ is replaced with a new random set of problems from B. Experiments are performed in which $|\mathcal{E}| \in \{100, 500, 1000\}$; at each iteration, the size of $\mathcal{H}$ and $\mathcal{P}$ are recorded. In order to demonstrate that the system has memory, the performance of the system against every potential problem in $\mathcal{U}'$ is tracked at each iteration. Particularly during early iterations, many of the problems in $\mathcal{U}'$ will not have been presented to the network therefore by measuring the hypothetical response against $\mathcal{U}'$, it is possible to gauge whether the system is generalising from seen instances and retaining that information. As $t \to \infty$, $\mathcal{E}^* \to \mathcal{U}'$.

–plotted both at every iteration (left-hand column) and averaged over each of the 1000 iterations the problems are present in $\mathcal{E}$ for. A number of trends are clear:

- The AIS can generalise over $\mathcal{U}'$; even in the early iterations we see good performance across the entirety of $\mathcal{U}'$ when only a small fraction $\mathcal{E}^*$ of it has been presented to the system.
- The system continuous to learn showing that knowledge of previously encountered problems is retained; the performance measured against all problems in $\mathcal{U}'$ improves over time; the rate of improvement can be increased by increasing the size of $\mathcal{E}$, the set of problems currently visible to the network; performance never deteriorates; the system therefore exhibits memory.
- Increasing $|\mathcal{E}|$, the number of problems in the environment, causes more difficulty at the start but has the effect of increasing the rate of learning overall. This is illustrated further in Figure 7.10 which summarise the results over 30 runs.

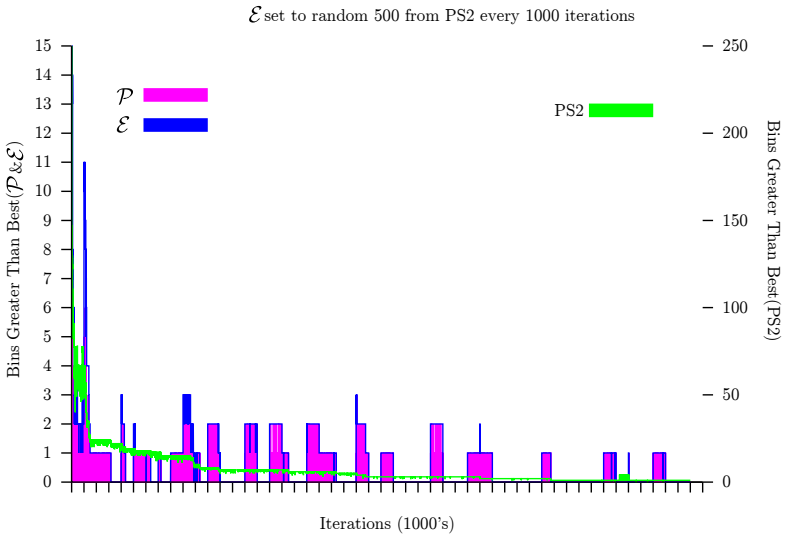


(a)

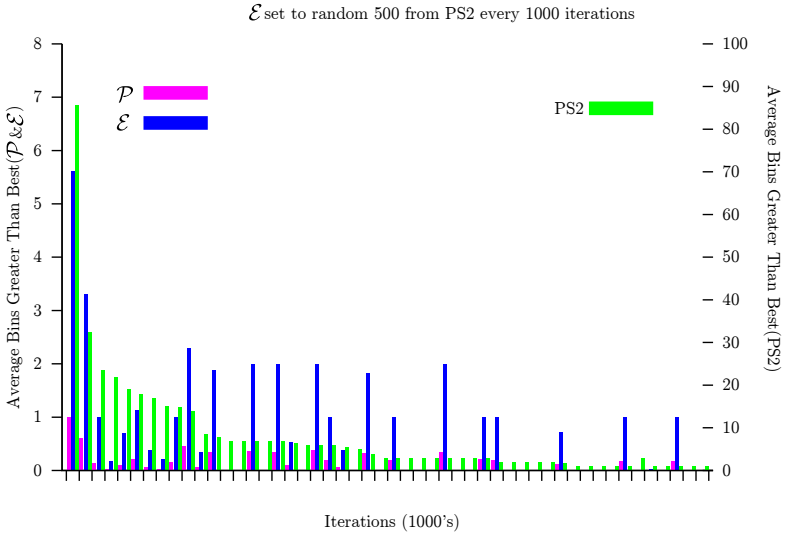


(b)

Figure 7.7: $\mathcal{E}$ changed every 1000 iterations to 100, 500 or 1000 problem instances randomly drawn from the full set of 3968 problem instances in problem set B. The plots show the number of bins greater than optimal plotted a) every iteration and b) averaged every 1000 iterations

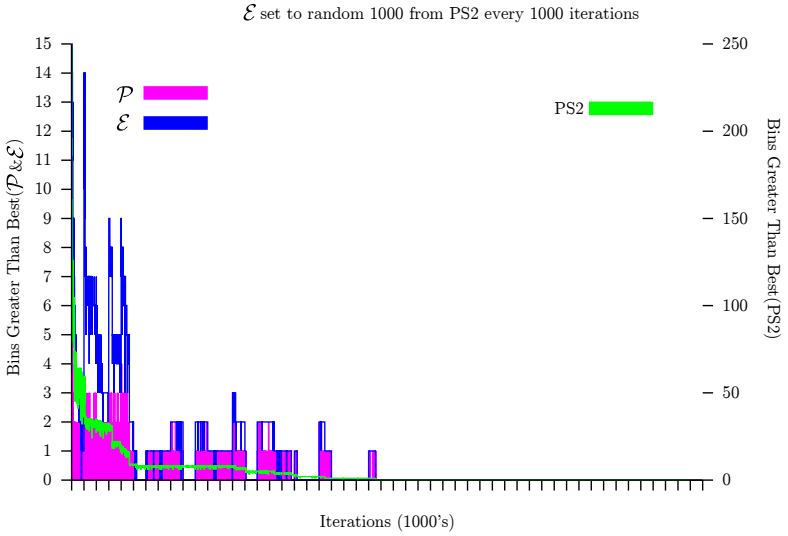


(a)

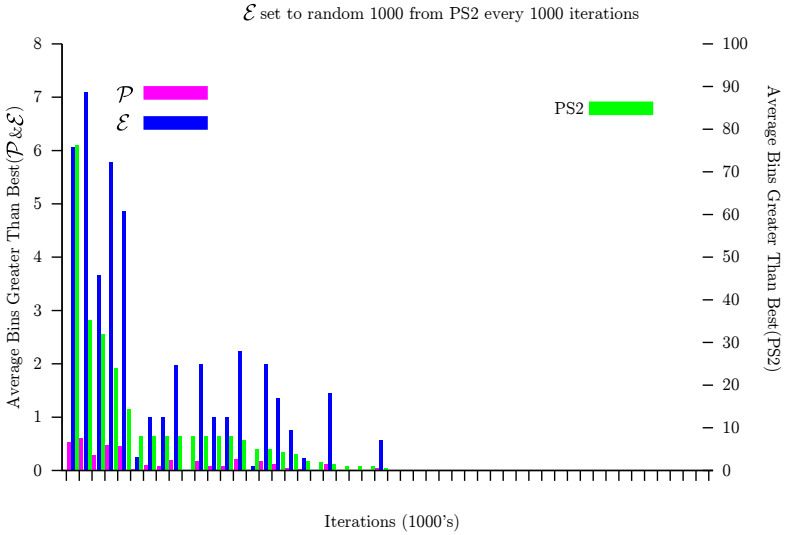


(b)

Figure 7.8: $\mathcal{E}$ changed every 1000 iterations to 500 problem instances randomly drawn from the full set of 3968 problem instances in problem set B. The plots show the number of bins greater than optimal plotted a) every iteration and b) averaged every 1000 iterations



(a)



(b)

Figure 7.9: $\mathcal{E}$ changed every 1000 iterations to 1000 problem instances randomly drawn from the full set of 3968 problem instances in problem set B. The plots show the number of bins greater than optimal plotted a) every iteration and b) averaged every 1000 iterations

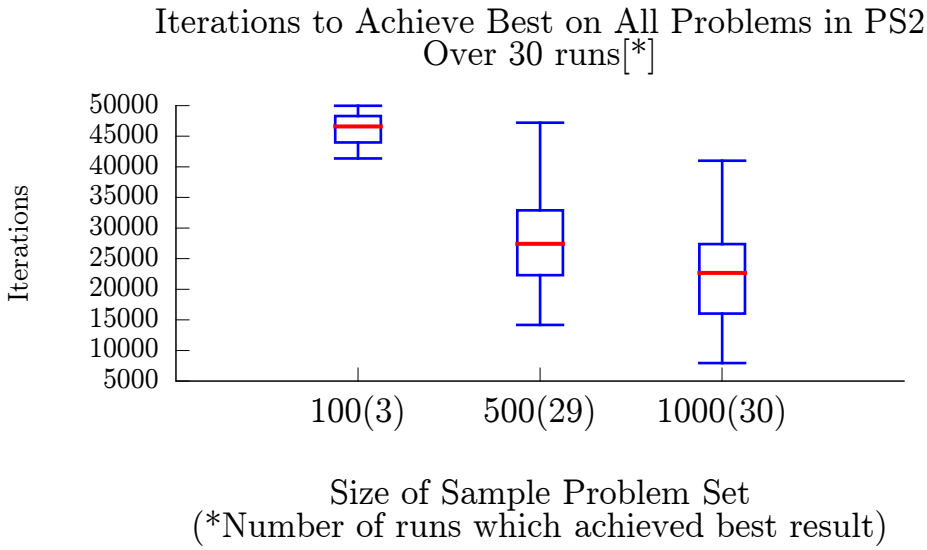


Figure 7.10: Number of Iterations to reach the best result for different sizes of $\mathcal{E}$

Memory and Learning: Response to new problems from different datasets

In order to demonstrate the systems learning and memory capabilities when faced with an environment in which problem characteristics vary over time, experiments are conducted using problems from B_{ds_1} and B_{ds_2}. These data sets — generated from parameters defined by [104] have been shown in previous chapters to have radically different properties. Heuristic that perform well on on the problem instances from ds_1 are unlikely will generalise to be the dominant heuristics when applied to the problem instances in ds_2.

In the following scenarios, the environment $\mathcal{E}$ is toggled alternately between B_{ds_1} and B_{ds_2} every 500 iterations. Two experiments were performed:

- The system was restarted every 500 iterations to obtain a benchmark response for the current set of problems presented (equivalent to a system with no memory)
- The problems in $\mathcal{E}$ were replaced every 500 iterations, but the heuristics present were retained (in order to test whether the system retains a useful memory)

In each of the two experiments, i.e. with and without memory, the number of extra bins required to solve the problems with respect to the best known solution using the

heuristics present in the network every iteration is calculated. Results are given in figure 7.11 which show the results over a single typical run. In these diagrams, the blocks alternate in colour to highlight the data-set being considered. The right-hand column is of most interest, as this shows the metric evaluated over $\mathcal{E}$, i.e. the set of problems in the system environment that we are currently interested in solving. The left-hand column of results represents the same metric but evaluated over $\mathcal{P}$, i.e. the set of problems that are sustained by the network as being representative of the problem space, and is shown to illustrate how the network is capable of generalising over $\mathcal{E}$ from the problems in $\mathcal{P}$. A number of observations can be made with regard to $\mathcal{E}$:

- The AIS — with its implicit memory — always outperforms the system with no memory. Due to the retained network, the system does not have to adapt from scratch to a new environment
- Adaptation still occurs in the system with memory, demonstrating the plasticity of the network
- Memory obtained during an epoch by running the AIS on a particular data set is sustained across subsequent epochs in which no items from that data-set are presented. This is apparent in the increasing performance on both data-sets over time.
- The problem instances in B_{ds_1} are clearly much easier than those from B_{ds_2} for the AIS: Within three presentations of samples from this data-set the system has reached optimal performance (i.e 0 bins greater than best) and sustains this performance for the duration of the experiment.[1]

Comparing the figures in the right-hand column to those on the left that represent the same metric evaluated over $\mathcal{P}$, we see that performance on $\mathcal{P}$ mirrors that of $\mathcal{E}$, i.e. an improvement on $\mathcal{P}$ correlates to an improvement in $\mathcal{E}$, confirming the generalisation capabilities of the network.

[1] Note that experiments showed that the order in which the two datasets are presented does not have any impact on the results.

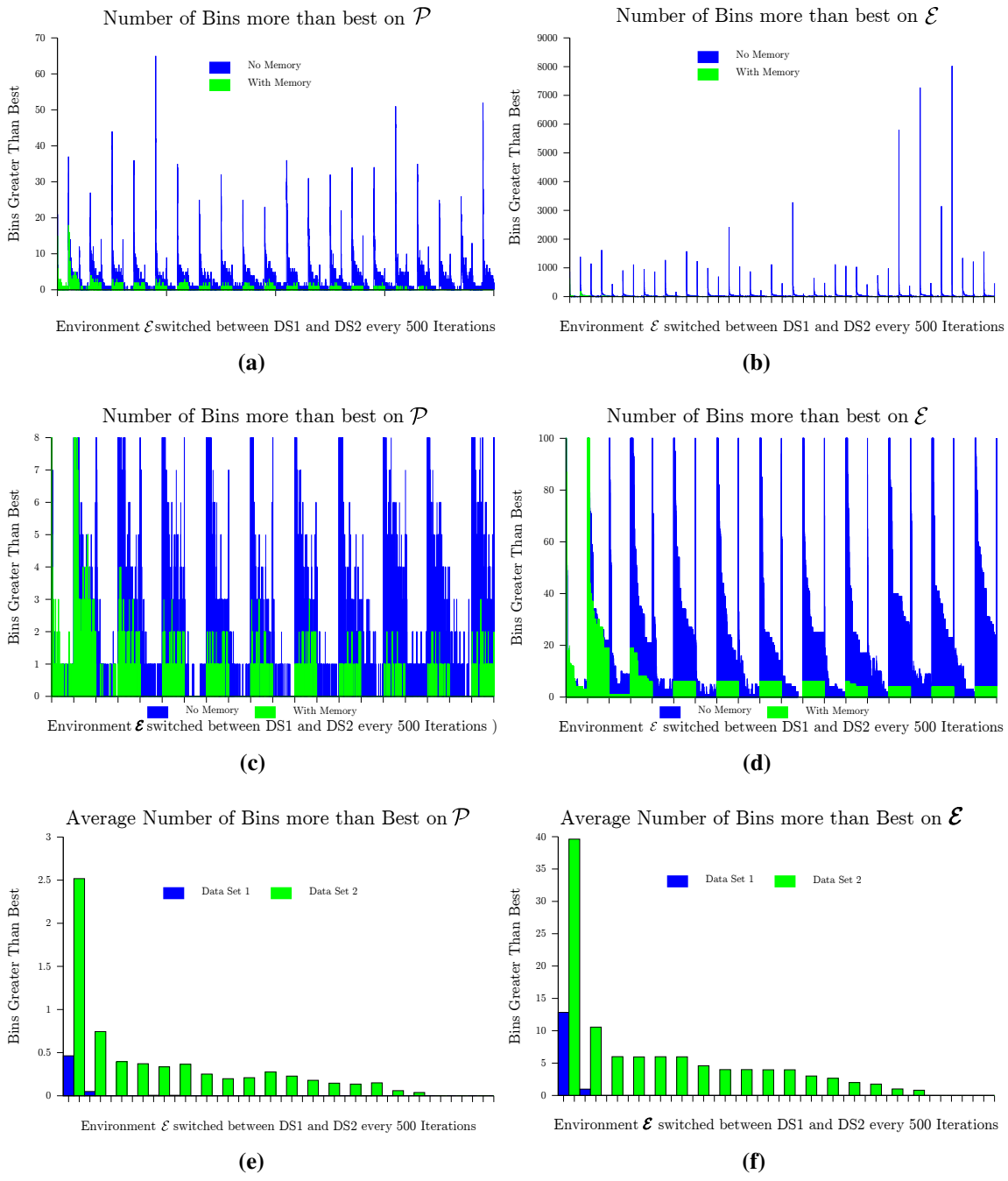


Figure 7.11: Alternating $\mathcal{E}$ between B_{ds1} and B_{ds2}. Utility measured against both $\mathcal{P}$ (left) and $\mathcal{E}$ (right). Figures 7.11c and 7.11c show an enlarged view of Figures 7.11a and 7.11b Figures 7.11e and 7.11f show the same results averaged over each epoch

7.6 Conclusions

This chapter has introduced an *artificial immune system hyper-heuristic* inspired by previous work in the artificial immune system field and the fields of generative and selective hyper-heuristics. The system is shown to have equal utility when compared to the island model introduced in the previous chapter with both systems consistently outperforming a range of human designed deterministic heuristics when evaluated on two large sets of problems totalling 5338 instances.

The AIS however exhibits number of advantages over the previously introduced island model.

- The AIS is plastic and highly responsive allowing it to adapt quickly to dynamically changing problem sets.
- The AIS retains memory using an efficient and novel method of summarising the problem space.

The system fuses methods from generative hyper-heuristics using SNGP to generate novel heuristics with ideas from immune-network theory, resulting in a self-sustaining interacting network of problems and heuristics that is capable of adapting over time as new knowledge is presented or if the environment changes.

The AIS was thoroughly tested in both static and dynamic environments using two test-suites comprising of a total of 5338 problem instances. The second of these test-suites (containing 3968 problems) was generated in order to provide a harder test than posed by existing benchmark problems; these problems are shown to be considerably more difficult than the standard benchmarks.

It is believed that the system described can be easily adapted to other combinatorial optimisation problem domains. In preliminary studies, published in [114], the system is adapted for application to the Job Shop Scheduling problem. The dynamics of the AIS remain identical to the system described here with only the heuristic generator modified to generate heuristics specific to the JSSP domain. Results show that by combining simple dispatching rules using the same methods outlined here that significantly improved results can be obtained when compared to the quality of the solution

attained by any of the individual hand-designed rules. Tested on a relatively small set of 62 JSSP instances the best evolved set of heuristics gives a total makespan of 65641 (3.6 % more than optimal) compare to the the best result obtained by a single dispatching rule which produced solutions with a total makespan of 71130 (12.3% more than the optimal of 63318).

The wealth of hyper-heuristics literature points to the utility of using multiple heuristics to solve problems. Specifically within that literature, there are many recent examples of Genetic Programming being used to evolve novel heuristics, for example [2] evolve heuristics for solving timetabling problems and [11] evolve new heuristics for solving 2D stock-cutting that provide promising avenues for future development.

Chapter 8

Conclusions

8.1 Overview

This chapter summarises the work presented in preceding chapters of this thesis and evaluates to what extent the original aims of the thesis have been fulfilled. The approaches documented in this thesis are discussed in relation to the work of others in the hyper-heuristic community and finally, some suggestions as to potential directions for further research are presented.

Hyper-heuristics have been touted to be general procedures capable of providing "good enough - soon enough - cheap enough" solutions to problems from different domains[14]. This study set out to look at the strengths of using hyper-heuristics to provide good solutions to problem instances from a wide range of different *classes* within the constraints of a single problem domain; the One Dimensional Bin Packing Problem Domain.

Three research questions were identified as outlined in Chapter 1 and repeated below for clarity.

- Question 1. To what extent can a deterministic constructive heuristic's ability for solving a problem be mapped to a problem's characteristics and therefore be exploited by a selective hyper-heuristic?
- Question 2. To what extent can novel heuristics be evolved that match or outperform human-designed deterministic constructive heuristics?

- Question 3. Can a hyper-heuristic be used to manage a collective of automatically generated heuristics that collaborate to efficiently cover large problem spaces composed of problems of differing characteristics?

The research presented in this thesis is limited to answering these questions within the constraint of a single problem domain. Furthermore the study is restricted to investigating these questions from the perspective of a subclass of hyper-heuristics; those that use deterministic constructive heuristics, both taken from the literature and automatically generated using Genetic Programming (GP) techniques. By limiting the study to the use of deterministic constructive heuristics more definitive conclusions can be drawn than would be possible if the uncertainty and unpredictability of perturbative heuristics were factored into the investigation. Although restricted in scope, it is the authors view that the conclusions drawn hold for hyper-heuristic approaches applied to any combinatorial problem where the objective is to optimise the permutation of items.

The study starts by identifying a mapping between the characteristics that best define a problem instance and the quality of the solution attained by each of four human designed heuristics. A hyper-heuristic is introduced that uses this correlation to exploit the combined utility gained by selecting intelligently from collectives of heuristics. Subsequent chapters show that individual constructive heuristics can be generated automatically which generalise better, over large problem sets, when contrasted to commonly used human designed heuristics. Furthermore sets of cooperative heuristics are generated which maximise their combined utility when applied to large problem sets containing a diverse range of problem instances of differing characteristics.

8.2 Defining a Mapping Between a Problems Characteristics and the Quality of the Solution Produced by a Heuristic

If a correlation can be determined between a problem instance's characteristics and the quality of the solution attained by a heuristic then it holds that the heuristic will

have varied performance when tested against large problem sets containing problems instances generated with a wide and diverse range of characteristics.

In Chapter 3 an investigative study concluded that although there were characteristics that affected the quality of the solution produced by a heuristic no single characteristic provided a suitable indication of this correlation in all cases. Furthermore, identifying which combination of characteristics were most relevant was not straightforward. Chapter 4 introduced a selective hyper-heuristic that included a learning mechanism that attempted to derive a combination of characteristics that when used as predictor attributes for a classification algorithm improved upon the correlation attained when compared to "'natural' human derived characteristics. The single characteristic which was found to have the largest correlation with the quality of the solution produced by a heuristic was the distribution of item sizes expressed as a ratio of the bin capacity. Figure 8.1 shows how the 4 deterministic heuristics perform in relation to this characteristic. The graph shows the average number of bins required over the known optimal plotted against the average number of items in an optimal solution.

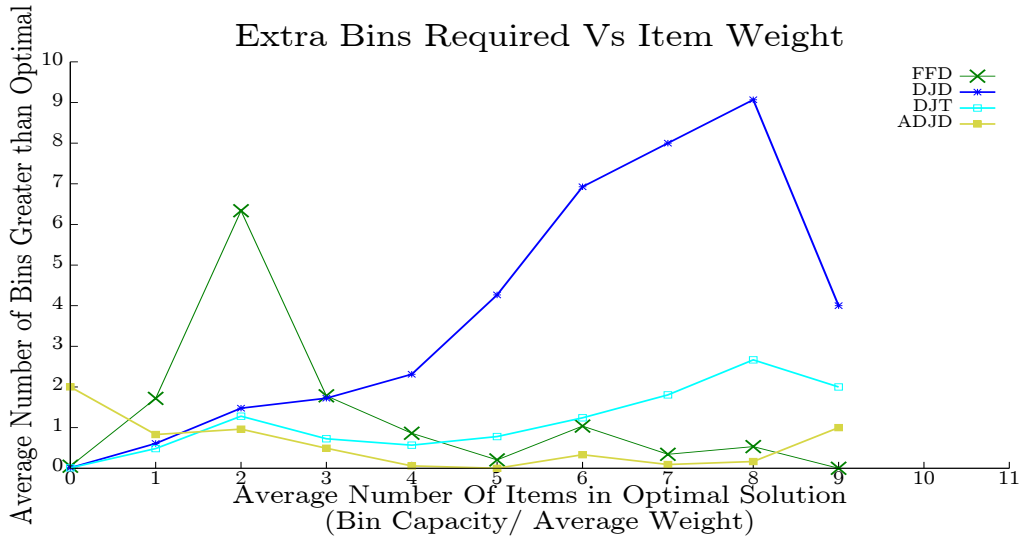


Figure 8.1: The performance of the human designed heuristics with respect to the average item weight. The graph plots the average number of bins more than the known optimal required by each heuristic against the average item weight expressed as a ratio of the bin capacity $(\frac{C}{\varpi})$

To improve classification accuracy this attribute was subdivided into a variable number and size of ranges with the quantity of each problem instance's items with sizes within the each range used as predictor attributes by the classification algorithm. An evolutionary algorithm was used to evolve the number and sizes of the ranges that best improved the classifier's prediction accuracy .

Once trained on half of a large set of 1370 problem instances, the hyper-heuristic exploits this correlation by applying the heuristic that is predicted to give the best result (selected from four human designed heuristics) for each of 685 unseen problem instances. By predicting the best from four heuristics to apply to each of a large unseen set of 685 problem instances the number of extra bins required over the optimal number was reduced by 51% (223) when compared to the solutions attained by the best of the four heuristics (DJT, 430).

Comparing the approach to the study that inspired it ([101]) the prediction accuracy was increased by 4% from 69% to 73% when contrasted with the accuracy attained using "natural" characteristics as predictors of solution quality. It is clear that the method introduced to classify problem instances is improved and is exploited by the selective hyper-heuristic leading to an overall improvement in solution quality compared to those solutions produced by any of the individual heuristics for all problem instances. However at around 73% accuracy there is room for improvement. The classifier attempts to predict which of the four heuristics will be best for each problem instance. The best heuristic was identified as the one that produced the solution that evaluated the highest using Falkenauer's fitness function. In order for the classification algorithm to have a chance to correctly predict the best of the four heuristics the solutions produced by each of the four heuristics should ideally be measurably different. However examining the solutions produced by the four heuristics shows that on some problems many of the solutions attained are structurally very similar. Many different solutions to the same problem instances cannot be distinguished if solution quality is measured using either the total number of bins required to pack all the items or by using Falkenauer's fitness function, which provides a higher degree of precision. This makes the job of the classifier more difficult as no clear winner can be identified in the case where different solutions are indistinguishable using either metric. During train-

ing if more than one heuristic obtained the best solution the classifier was told that the simplest (in terms of computational effort) of the winning heuristics was the best in an attempt to increase the efficiency of the system. Ultimately the system is restricted by two factors; the ease with which many of the heuristics solve many of the benchmark problems and by the similarity of the solutions produced by the heuristics on certain portions of the problem space.

Although there are many documented heuristics in the literature that can be used to solve the BPP, the quantity and more importantly the lack of diversity of human designed heuristics proves restrictive to the selective hyper-heuristic introduced. For a selective hyper-heuristic approach to be effective it must be provided with a set of diverse problem specific heuristics over which to search. If the behaviour of the underlying heuristics is too similar or they are tailored to perform well on small portions of the problem space then the effectiveness of the hyper-heuristic will be compromised. At the start of the period of study a number of other heuristics were investigated and subsequently disregarded due to the similarity of the solutions produced when compared to those attained by other heuristics. For example the Best Fit Descending heuristic gives solutions to 1369 of the 1370 problem instances in Problem Set A using in each case the same number of bins as the solutions attained using FFD. On the other problem instance it improves on the solution provided by FFD by a single bin.

The second factor that restricts the ability of the classifier is the simplicity with which optimal solutions are found to many of the benchmark problem instances used throughout this study. Many of these problem instances have been found to be easily solved using simple heuristics or even manual methods [57] and identifying the minor nuances in individual problem instance's characteristics that cause them to be more (or less) difficult for a particular heuristic is not always possible. Of the 1370 problem instances in Problem Set A, 187 are solved using the optimal number of bins by all 4 of the man made heuristics. A further 824 are solved optimally by more than one of the heuristics and a further 79 are solved optimally by a single heuristic. Given that the heuristics are all simple deterministic methods it could be argued that these benchmark problems are not particularly taxing for the simple heuristic methods. The ability to predict the best heuristic to apply for many of these instances is unnecessary

and unproductive as the optimal solutions are easily attained and the similarity of the solutions does not allow the classifier to learn associations which may exist.

There is clearly a benefit to selecting the best heuristic from a collective of heuristics but in order to maximise this potential the set of heuristics must provide significantly different solutions to individual problem instances and uniquely provide the best solutions on niche areas of the problem space.

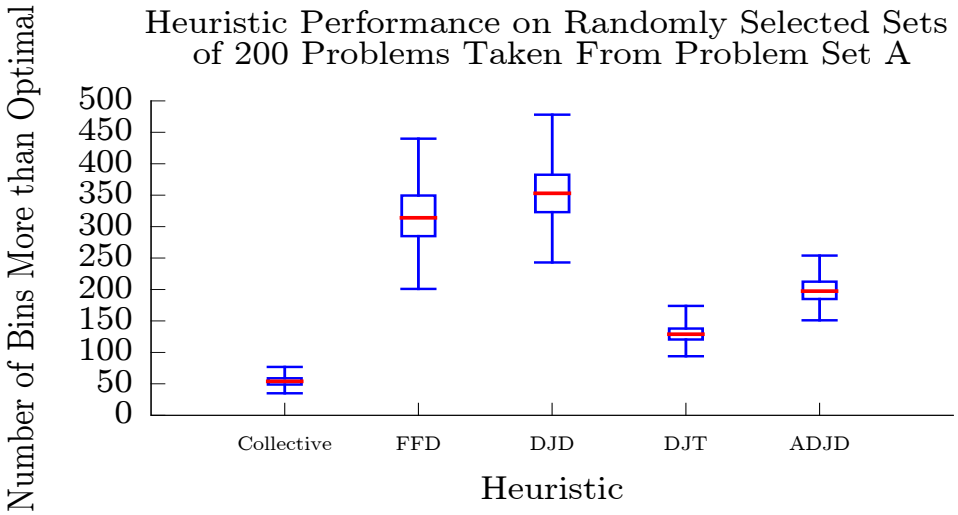


Figure 8.2: The performance of the individual human designed heuristics compared to their combined utility. Each box plot shows for a different 100 sets of 200 problems randomly selected from the 1370 in Problem Set A the number of bins more than the optimal required by each heuristic. The box plot marked collective shows how a greedy selection of the 4 heuristics performs on the same 100 sets of 200 problems

Figure 8.2 highlights the maximum possible improvement to be gleaned by exploiting the combined ability of the four deterministic heuristics. For this plot 100 different sets of 200 problem instances were selected at random from the 1370 problem instances included in Problem Set A. Each of the 4 heuristics were used to solve each of the 200 problems and the number of extra bins needed over the known optimal plotted. Contrasted against the utility of the 4 individual heuristics the box plot labelled *Collective* shows the combined utility of the 4 heuristics if the best heuristic was selected greedily for each of the 100 sets of 200 problem instances. It is clear, and

to be expected, that the collective performance is greater than that of the individual heuristics. Interestingly the deviation in results attained by the collective of heuristics is minimal when compared to any of the individual heuristics which can have varying performance on certain problem instances with certain characteristics.

The classification mechanism used in the hyper-heuristic introduced in Chapter 4, predicts, selects and applies one heuristic for each problem instance based on knowledge learned during an off-line training phase. Although this training phase is computationally expensive it is only required to be conducted once after which the resultant algorithm can be quickly applied to an unseen instance in order to to decide which heuristic to apply. Obviously the best result that could be produced by predicting the best heuristic would be the same as selecting from the four heuristics using a greedy strategy (which yields 60,445 bins or an improvement of 35 bins over the result obtained by the selective hyper-heuristic) In this study applying all 4 heuristics and selecting the best solution is trivial and relatively quick but as the problem space and the number, or complexity, of the heuristics used increases the task would become more computationally expensive and using intelligent selection strategies could help to alleviate this.

8.3 Generating Novel Heuristics for the BPP

In an attempt to increase the variety and utility of the constructive heuristics available to a selective hyper-heuristic a number of generative approaches were used to automate the design of constructive deterministic heuristics for the BPP. Single heuristics were generated that generalised well over large problem sets and collectives of heuristics were co-evolved that collectively improved upon the performance of any of the individual heuristics. All of the hyper-heuristics implemented generate heuristics by combining the nodes described in Table 5.1 using a form of GP (Single Node Genetic Programming, SNGP). Each generated deterministic heuristic is used in isolation to iteratively pack bins until all of the items are placed into a solution.

8.3.1 Generating Single Heuristics

In Chapter 5 SNGP was used to generate individual heuristics that were trained and tested on equal divisions of the 1370 problems in Problem Set A. The results presented in Table 5.3 and Table 5.4 clearly demonstrate the benefits of automating the process of designing heuristics. When compared to best results obtained by any of the benchmark human designed heuristics on the full set of problems the the best evolved heuristic (trained on the training set) managed to find optimal solutions to 19% more problem instances (1028) and solved all problem instances using 40% fewer extra bins (550) over the optimal (120433). The generated heuristic managed to solve 93% of the 1370 problem instances using no more than 1 bin more than the known optimal and in the worst case (1 problem instance) required 9 bins more than the known optimal. In contrast FFD and DJD required 10 bins or more to solve 50 and 60 of the problem instances respectively.

Although significantly improved results are attained when compared to any of the individual human designed heuristics, using a greedy selection strategy to select the best from the pool of 4 human designed heuristics still yields further benefits (1090 problems solved optimally and all problems solved using 364 extra bins than the known optimal). The single best evolved heuristic generalises well over the complete set of problem instances that it was trained on and has better worst case performance on these instances than any of the human designed heuristics. However, it is less successful than individual human designed heuristics when evaluated on niche areas of the problem space and consequently selecting greedily from the human designed heuristics yields benefits when evaluated across the complete problem set.

As noted by Burke et al. [16] it is possible to either generate specialised heuristics that perform well on small sets of problems with similar characteristics or general heuristics that provide a good performance on larger more diverse sets of problems. In order to improve upon the result achieved by any single heuristic it is necessary to generate sets of heuristics that specialise to cover different areas of the problem space yet collectively generalise over the complete landscape.

8.3.2 Generating Collectives of Cooperative Heuristics

Chapters 6 and 7 introduce two different approaches that are used to generate sets of heuristics that cooperate to maximise their collective performance over the complete set of problem instances that they were tested on. An evolutionary island model is presented in Chapter 6 and an Artificial Immune inspired approach is introduced in Chapter 7. Both systems incorporate multiple SNGP implementations to concurrently evolve sets of heuristics that collectively cooperate to cover the problem space. Both systems were trained and tested on the same division of problem instances from Problem Set A that were used to train and test previously described hyper-heuristic approaches.

Performance Gained from using Collectives of Heuristics

The combined performance of the best co-evolved set of heuristics, for both systems, increases the number of problems solved optimally to 1126 and reduces the number of extra bins required to 308. In comparison the best single evolved heuristic required 550 bins extra and the best human designed heuristic required 881. The evolved set of heuristics also performs better than the collective of four human designed heuristics which requires 364 bins more than the optimal number when evaluated across all 1370 problems.

The results presented for both the island model and the immune model are virtually identical. Both systems were evaluated over 30 runs and both solved all of the problem instances in Problem Set A using at best the same number of extra bins. The mean number of extra bins was also 308 for both models with the island model having a marginally higher standard deviation; 1.4 compared to 0.3 for the immune inspired algorithm. Both systems use the same Single Node Genetic Programming (SNGP) model to generate heuristics using identical sets of function and terminal nodes as heuristic components that were identified by manually de-constructing the benchmark heuristics into there component parts. It is therefore not surprising that the sets of heuristics generated by both systems have equal performance when evaluated over the same sets of problem instances.

Implicit Mapping to Problem Characteristics

Although the collectives of heuristics consistently produced the same results these were not always attained using the same number of heuristics which varied between 6 and 8 for both systems. In both systems there was no limit on the maximum number of heuristics that were allowed in the system. The number of heuristics sustained provides a measure of how well the heuristics can co-evolve to cover different areas of the problem space as well as giving an indication of the diversity of the underlying problem space from the perspective of those heuristics.

The areas of the problem space that cause the heuristics be sustained suggest those niche areas of the problem space that require exploring and conversely the areas that prove relatively easy for the heuristics can be identified and avoided. Rather than attempting to explicitly derive a mapping between heuristics and the characteristics of the problem instances that they operate best on, the evolved heuristics are sustained by the system only if they uniquely provided the best solution to niche areas of the problem space. The dynamics of both cooperative approaches implicitly sustains heuristics that operate best on different portions of the problem space as illustrated in the example in Figure 8.3. The ability to generate heuristics that specialise to different areas of the problem space is limited by the ease with which many of the problems are solved and the limited utility of the heuristics that can be created from the components identified. It is interesting to note that while collectively the evolved sets of heuristics outperform the single best evolved heuristic, none of the heuristics that make up the collective gets near to the performance of the single evolved heuristic when evaluated in isolation over the complete set of problems.

During training the quality of the solutions attained by the heuristics (as gauged by the number of bins required) is used only as a comparison of how well each heuristic performs in relation to the other heuristics in the system. It is not used as an indication of how close to optimal that solution is. In fact if solutions using the optimal number of bins are obtained to any problem instances by more than one heuristic the dynamics of both of the systems disregards any influence from those problem instances. Evolutionary pressure is exerted by newly introduced heuristics which can only survive

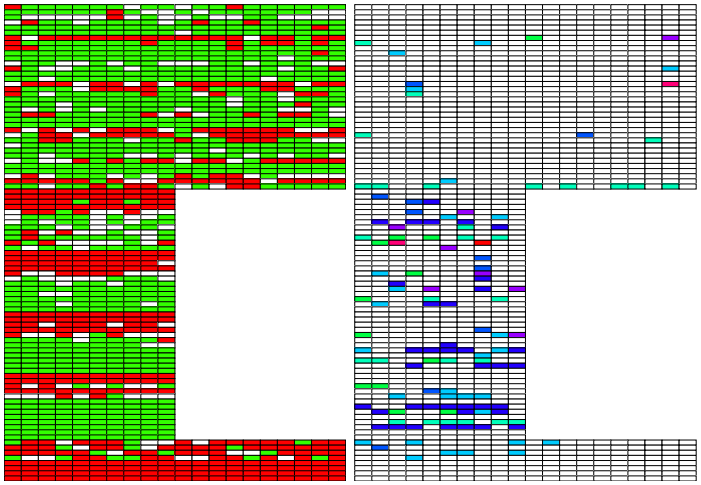


Figure 8.3: Each cell represents a different problem instance from Problem Set A. Each row represents a sequence of problem instances that were all generated using the same parameter settings. The colours represent different heuristics. Coloured cells indicate those problem instances that the heuristic produces the *best* solution for. White cells indicate that those problem instances are solved *equally well* by more than one heuristic in the system. The diagram on the left shows how two heuristics seperate to cover distinct areas of the problem space. On the right the number of heuristics is unrestricted and many problem instances are solved equally well by more than one heuristic.

if they are able to outperform all of the heuristics currently in the system on a niche area of the problem space. If two heuristics or more provide equally good solutions then the dynamics of the system will eliminate all but one. The effect of this is that all of the heuristics sustained by the system will be *best* on at least some of the problem space and collectively the set of heuristics sustained will have better performance than any of the individual heuristics. The dynamics also favours sets of heuristics of lower cardinality. If a single heuristic is injected into the system which matches (or exceeds) the quality of the solutions obtained to the problem instances currently solved best by two other heuristics then the two heuristics will be replaced. This has the effect of minimising the repertoire of heuristics required to best cover the complete problem space thereby improving the efficiency of the system.

Trade off Between Adding more Heuristics and the Collective Improvement in Solution Quality

As the number of heuristics sustained increases the probability of generating further heuristics that are able to specialise and populate niche areas of the problem space becomes more difficult. In Figure 8.3 the diagram on the left shows an example of how two randomly generated heuristics diversify to find problem instances that they solve best. In this example there are few problem instances that are solved equally well by both heuristics (white cells). If the number of heuristics injected is unrestricted then the the search space is quickly saturated and consequently the number of problem instances that are solved equally well by more than one heuristics grows rapidly as is illustrated by the right hand diagram which shows a collective of 6 heuristics and how they cover the problem space.

When the number of the problem instances presented to the system is increased the number of heuristics sustained by the system also increases. This is shown in the analysis provided in Chapter 7 where the AIS was applied to the much larger Problem Set C totalling 15,830 problem instances. These problem instances were selected from a much larger set of newly generated problem instances as potentially interesting due to the fact that none of the human designed heuristics were able to find any optimal solutions. Although these problems were generated using the same parameter settings as many other more easily solvable problems there are clearly nuances that affect their complexity from the perspective of the benchmark heuristics. An interesting feature of these *hard* problem instances is that the ability of the heuristics, both generated and human designed, to find a solution requiring only one bin more than the optimal seems unaffected when compared to the problems in Problem Set A. This indicates the likelihood of many plateaus in the search space of slightly less than optimal quality, a feature that could be exploited by a hyper-heuristic aiming to find quick acceptable solutions to these problems.

The techniques described here aim to achieve the best results possible but there is a trade off between the benefit of adding more heuristics and the computational resources required. Both the island model and the AIS can easily be limited to a maximum num-

ber of heuristics rather than being unrestricted. The benefit of adding extra heuristics to the collective performance quickly decreases as is shown below in Figure 8.4. This figure shows the improvement in performance as more heuristics are added and greedily applied to the problem instances in Problem Set A. When the AIS is applied to the problem instances from problem set C which are generated with similar characteristics to those in Problem Set A the number of heuristics sustained is increased from between 6-8 to over 20. However for different sizes of benchmark problem sets the number of heuristics required to cover the problem space remains almost constant if measured as a ratio of the number of problem instances included. The graphs shown in Figure 7.6 further highlight the trade off between adding more heuristics and the collective improvement in solution quality. It is interesting to note that after an initial period of fluctuation the number of heuristics sustained remains almost constant but those heuristics continue to improve upon their collective utility for many more iterations.

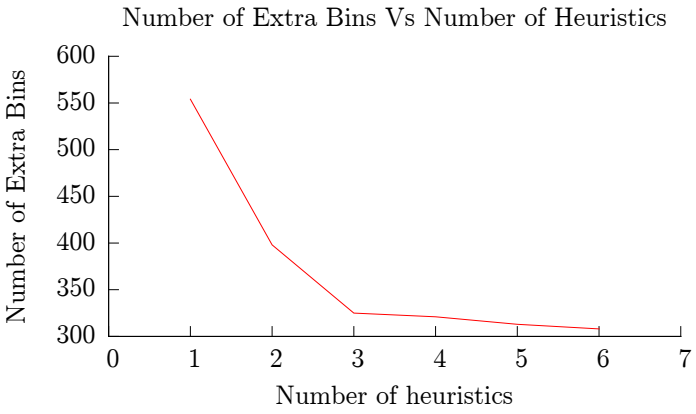


Figure 8.4: As more heuristics are added the benefit in terms of the combined performance becomes smaller.

Minimising the Weaknesses Inherent in Individual Heuristics

Many of the results presented here measure the utility of the hyper-heuristics in relation to the one of the most commonly used metrics for the BPP, namely how many problems were solved optimally. While this is undoubtedly a valid metric by which

to gauge the success of a particular method it is not in keeping the original motivating goal behind hyper-heuristics which need only provide adequate solutions in a timely manner to a broad range of problem classes. While individual human designed heuristics show particularly bad performance on certain problem instances the collectives of heuristics generated in chapters 6 and 7 cooperate to minimise the individual heuristics weaknesses. Table 7.4 shows that 97% of problem instances in Problem Set A are solved using at most 1 extra bin than the known optimal and that at worst 2 problem instances require 5 extra bins. In contrast the commonly used first fit descending heuristic requires 10 or more extra bins for 50 problem instances and up to 25 extra bins in the worst case. The collective of co-evolved heuristics appear to exploit the many plateaus in the search space of slightly less than optimal quality that individual heuristics cannot. As noted previously the best single evolved heuristic still required up to 9 extra bins for some problem instances.

Benefits Associated with the AIS compared to the Island Model

The immune inspired approach has a number of advantages over the island model in terms of the ease with which it can adapt to unseen problem instances and in terms of its efficiency. The knowledge that many heuristics provide solutions that are immeasurably different[1] for many problem instances is used to minimise the set of problem instances used during training thereby increasing the efficiency of the system. Only problem instances that are solved best by a single heuristic are retained resulting in a minimal repertoire of heuristics that individually contribute to the combined utility of the collective and a minimal repertoire of problem instances that describe the problem space from the perspective of those heuristics.

The AIS achieves the same results as the island model but is significantly more efficient as the heuristics only have to be applied to a small representative set of problem instances that describe the complete problem space. Although the final solutions produced by the collective of heuristics remain virtually unchanged across 30 separate

[1]The solutions produced may have slight structural differences but these cannot be distinguished using either the number of bins or Falkenauer's fitness function to evaluate them.

runs the set of heuristics and problem instances that are sustained changes. The number of problem instances sustained by the system when applied to the 1370 instances in Problem Set A varies between 26 and 56 and the number of heuristics fluctuates between 6 and 8 (as it did with the Island model).

Unlike the island model the heuristics co-evolved by the AIS are not created in isolation but are incorporated into the same evolutionary system that does not converge in the same manner that a conventional EA does but maintains diversity as an integral part of the system. The study presented in Chapter 7 shows the benefit of such an approach in a constantly changing environment where the set of problem instances presented to the system is constantly changing. The AIS is shown to provide good solutions quickly yet is shown to continually improve on the utility of the system.

8.4 Summary

The hyper-heuristics presented in this thesis, when evaluated across large problem sets, are more successful than their human designed counterparts either when evaluated individually or as part of a collective. This study set out to investigate whether hyper-heuristics could provide adequate solutions to a range of different problem instances with widely different characteristics within the BPP domain. Table 8.1 summarises the incremental improvement in solution quality presented during the different chapters of this thesis.

Hyper-heuristics have the potential to provide simple understandable procedures for obtaining good solutions to a range of problems of different characteristics. Simple heuristics can easily be adapted to cope with the constraints imposed on real world problems that both complicate and limit the use of more traditional stochastic search techniques by industry. Furthermore by generating collectives of heuristics that autonomously diverge to solve niche areas of the problem space the utility of any hyper-heuristic can be maximised.

While it is clear that there exists a relationship between the utility of a heuristic and the problem instances that the heuristic is best suited to solving it is difficult to quantify and will vary depending on the problem space and the behaviour of the particular

Table 8.1: Comparison of benchmark heuristics and all hyper-heuristics implemented on problem set A

Heuristic(s)	Problems Solved	Extra Bins Required
FFD	788	2142
DJD	716	2409
DJT	863	881
ADJD	686	1352
Collective	1090	364
Selective HH	1044	429
Single H (Generated)	1028	550
Island	1126	308
AIS	1126	308

heuristic. Furthermore the ability to generalise about this correlation when collectives of heuristics are used becomes more difficult as the number of heuristics and problem instances are increased. The co-operative approaches presented in later chapters of this thesis alleviate these difficulties by explicitly evolving heuristics that specialise on problem instances in niche areas of the problem space. The hyper-heuristics presented in this thesis only cover a subclass of the many different hyper-heuristic approaches documented. The following section briefly evaluates some of the more successful hyper-heuristic approaches that have recently emerged in the literature that have been applied to the BPP.

8.5 Other Recent Hyper-heuristic Approaches for the BPP

The research presented in this thesis is restricted in scope to investigating hyper-heuristic approaches that utilise deterministic constructive heuristics specific to the off-line variant of the One Dimensional Bin Packing Problem (BPP). Limiting the study to the use of deterministic heuristics allowed definitive conclusions to be drawn

when comparing the approaches introduced with other deterministic heuristics taken from the literature. Hyper-heuristics cover a broad range of methods that utilise a large number of different techniques of which amongst the most successful approaches in the literature for the BPP are those that utilise perturbative heuristics. Rather than building a solution from scratch perturbative hyper-heuristics iteratively apply one or more neighbourhood move operators to an incumbent solution.

As an example, in [20] conventional GP is used to evolve heuristics that are tested on 90 problem instances from Problem Set A. The results presented are just less than optimal on the problem instances used for testing. In the work presented in [22] the authors evolve local search heuristics using grammatical evolution that are applied to 70 of the problem instances from Problem Set A. Again the results achieved are only marginally worse than optimal on these problem instances. While both of these studies report better results than any of the hyper-heuristics presented in this thesis they are only evaluated on much smaller sets of problem instances. Both approaches could be criticised as digressing from the ideology behind hyper-heuristics in that they are used to generate a throwaway heuristic for each problem instance, often over many thousands of evaluations. For example in [20] a new heuristic is generated for each problem instance using GP to combine problem specific components. Each evolved heuristic is created over the course of 50,000 function evaluations. In both studies the set of problem instances used is small ranging from 70 - 90. Furthermore the majority of the problem instances are very similar with 80 of the 90 problem instances used for testing in [20] having item weights drawn from the same distribution. Neither of these approaches could not be described as being quick and although the solutions produced are almost optimal the methods used could be criticised as having the same drawbacks as many metaheuristic techniques which are often computationally expensive and offer no degree of confidence in the solutions provided due to their stochastic nature. Hyper-heuristic approaches are intended to provide "good enough" solutions to problem instances in a timely manner without needing to be reconfigured when presented with unseen problem instances of varied characteristics.

All of the heuristics used in this thesis, either human designed or automatically generated are simple reusable procedures that provide solutions to most problem in-

stances within a few milliseconds. Although on occasion the solutions produced are less than optimal they are significantly improved upon the solutions attained by any of the individual human designed heuristics that they are compared to. Burke et al. [16] conclude that it is possible to either generate specialised heuristics that perform well on small sets of problems with similar characteristics or general heuristics that provide a good performance on larger more diverse sets of problems. The research presented here backs up this assertion but shows that it is possible to generate sets of heuristics that individually specialise to niche areas of the problem space yet collectively generalise over larger sets of problems without the associated reduction in performance inherent in any individual heuristic when applied to increasingly large problem sets.

8.6 Future Work

A number of potentially interesting avenues for future research are discussed in the remainder of this chapter.

Granularity of Heuristic Components

The nodes used to generate heuristics in this study are relatively coarse grained and can even be used as complete heuristics in there own right. In both of the hyper-heuristics described in this thesis that were used to to generate sets of heuristics the number of heuristics sustained is directly correlated to the set of problem instances that they were used to solve and the granularity of the nodes used to construct heuristics. It may be possible to implement hyper-heuristic approaches that specialise to individual problem instances using finer grained components but the trade off in computational complexity needs to be offset against the improvement in solution quality.

Combining Constructive Heuristics

A limitation of the hyper-heuristics presented in this thesis is that for any problem instance a single heuristic is used to construct a solution. In contrast the work that in-

spired the selective hyper-heuristic introduced in Chapter 4 attempts to select different constructive heuristics during the process of solving a single problem instance [101]. The authors were able to find optimal solutions to certain problem instances that none of the underlying heuristics were able to produce if used in isolation. All of the heuristics generated in this thesis, as well as the human designed heuristics that were taken from the literature, were designed to pack a single bin at a time. By using different heuristics to pack individual bins for a single problem instance it may be possible to improve upon their standalone performance.

Combining Generative and Selective Hyper-heuristics

Hyper-heuristics can be classified as belonging to one of two subclasses; Generative hyper-heuristics or Selective hyper-heuristics. Both of the hyper-heuristics introduced in Chapters 6 and 7 partially combine these methodologies by selecting from sets of automatically generated heuristics, those that cooperate to best solve large sets of problem instances of wide ranging characteristics. Although heuristics are co-evolved to work on niche areas of the problem space ultimately the selection mechanism used is greedy and although it is relatively quick to apply each of the small set of 6 heuristics sustained to a new problem instance, the technique would become more laborious if the number and diversity of the problem instances was increased to such a level that many more heuristics were needed to cover the problem space.

Chapter 4 showed to a large degree that a mapping could be determined between individual heuristics and the problem instances which they will work well on. Further research is required to improve upon this classification technique if it were to be applied to larger sets of heuristics which would pose more of a challenge for the classification algorithm as the number of heuristics sustained by the system is increased. Combining the utility of different heuristics on different parts of the problem space has been shown to be highly advantageous when contrasted to the ability of a range of human designed heuristics. However the improvement in solution quality attained by the collectives of heuristics introduced in Chapters 6 and 7 over the best single heuristic evolved in Chapter 5 is not as significant as may be expected and there is an obvious

trade off between the performance gain to be attained by introducing more heuristics and the computational resources required to generate and apply these heuristics. On larger and more diverse problem sets, or in the case where the heuristics were created from finer grained nodes this method of selecting from the set of heuristics that are sustained may prove too costly.

Improving the Immune Model

The immune inspired hyper-heuristic introduced in Chapter 7 minimises the number of evaluations required by summarising the problem space using only those problem instances which are solved best by any single heuristic. This is shown to have no detrimental affect on the overall utility of the system but to dramatically reduce the computational effort required. The thesis set out to find a mapping between a particular heuristic and the performance on a given problem instance. Future research could concentrate on refining the model and the affinity metric used in the AIS presented in Chapter 7 to better fit with the immunological metaphor in order to exploit the affinity that exists between the heuristics and problem instances sustained by the system.

Combining Constructive Heuristics with Improvement Heuristics.

Another potential avenue for future research is in the combined use of both constructive and local search heuristics within a single framework. Most perturbative hyper-heuristics work by iteratively applying local move operators to complete solutions that are either initialised randomly or using simple constructive heuristics. In contrast constructive hyper-heuristics apply one or more heuristics in turn to iteratively build a solution. There may be potential in combining these approaches by applying perturbative approaches to a more varied set of solutions initialised using sets of automatically generated constructive heuristics. Research in the metaheuristic community has shown such hybrid methods to be amongst the most successful for solving combinatorial problems.

Understanding the Benefits to be Gained by Utilising Sets of Weak Heuristics.

The utility to be gained from combining automatically generated heuristics is shown empirically in Chapters 6 and 7. In both chapters the individual heuristics that make up each collective are relatively weak in comparison with either the human designed heuristics examined in this thesis or the best single heuristic evolved in Chapter 5. In the optimisation community it is well known that no individual algorithm can outperform all others when applied to large diverse data sets containing many diverse problem classes. There is a trade off between the ability of any individual heuristic which attempts to generalise across the complete problem space and the computational effort required to search for sets of more specialised heuristics which are individually suited to niche areas of the problem space but perform poorly when evaluated on larger more diverse problem sets.

Theoretical studies in the machine-learning community have highlighted similar benefits when combining classifiers such as is the case with Bootstrap Aggregating or *bagging* [39] where multiple classifiers are used in order to alleviate any weaknesses shown by individual classifiers. Other examples from the machine learning community include for example [68] where multiple perceptrons, which individually were shown be little better than random guessing when applied to classification tasks, were combined and shown to collectively outperform more sophisticated classifiers. The hyper-heuristic community would benefit from a better understanding of algorithm behaviour that could in part be driven by the work conducted in the area of machine learning and could potentially lead to a better understanding of algorithm design.

Closing Remarks

Hyper-heuristics may not be the most glamorous search mechanisms to be introduced when contrasted to more conventional search methods that claim to offer optimal solutions to many academic problems. The approaches presented in this thesis are however relatively simple and quick to execute and are shown to provide almost optimal solution for most of the problem instances that they were evaluated on. In order to ap-

ply a particular search technique to real world problems the techniques must be simple enough for non experts to understand, provide a level of assurance in the quality of the solution that they provide and be extensible enough that they can be modified to cope with the many constraints that are commonly not factored into contrived benchmark problems. For many real world combinatorial problems, heuristics are potentially the only realistic methods of generating solutions where complete methods are infeasibly slow and metaheuristics methods are too complex to be easily adapted to deal with application specific constraints and the requirements of industry. By combining simple and easy to understand heuristics the potential to provide good solutions to many industrial problems without the complexity and uncertainty associated with more conventional search techniques makes future research into hyper-heuristics a useful and potentially rewarding avenue for further study.

Bibliography

[1] Sam Allen, Edmund K. Burke, Matthew Hyde, and Graham Kendall. Evolving reusable 3d packing heuristics with genetic programming. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 931–938, New York, NY, USA, 2009. ACM.

[2] M. Bader, R. Poli, and S. Fatima. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, 1(3), 2009.

[3] Mohamed Bader-El-Den and Riccardo Poli. Generating sat local-search heuristics using a gp hyper-heuristic framework. In *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 37–49. Springer Berlin Heidelberg, 2008.

[4] R. Bai, E. K. Burke, M. Gendreau, G. Kendall, and B. McCollum. Memory length in hyper-heuristics: An empirical study. In *Computational Intelligence in Scheduling, 2007. SCIS '07. IEEE Symposium on*, pages 173–178, 2007.

[5] Ruibin Bai and Graham Kendall. An investigation of automated planograms using a simulated annealing based hyper-heuristic. In Ramesh Sharda, Toshihide Ibaraki, Koji Nonobe, and Mutsunori Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, volume 32 of *Operations Research/Computer Science Interfaces*, pages 87–108. Springer US, 2005.

[6] Ruibin Bai, Jacek Blazewicz, EdmundK. Burke, Graham Kendall, and Barry

McCollum. A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR*, 10(1):43–66, 2012.

[7] J E Beasley. Or-library. Retrieved March 2011, from http://people.brunel.ac.uk/ mastjjb/jeb/info.html, 1990.

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

[9] E. Burke, M. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 860–869. Springer Berlin / Heidelberg, 2006.

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

[11] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward. A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942 –958, 2010.

[12] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyper-heuristics: a survey of the state of the art. *J Oper Res Soc*, 64 (12):1695–1724, Dec 2013.

[13] Edmund Burke and Naimah Hussin. *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, chapter A Tabu Search Hyper-heuristic Approach to the Examination Timetabling Problem at the MARA University of Technology, pages 270–293. Springer Berlin / Heidelberg, 2005.

[14] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search

technology. In *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, chapter 16, pages 457–474. Kluwer, 2003.

[15] Edmund K. Burke, Matthew Hyde, Graham Kendall, and John Woodward. Scalability of evolved on line bin packing heuristics. In *IEEE Congress on Evolutionary Computation (CEC07)*, pages 2530–2537. Singapore, September 25-28 2007.

[16] Edmund K. Burke, Matthew R. Hyde, Graham Kendall, and John Woodward. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1559–1565, New York, NY, USA, 2007. ACM.

[17] 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.

[18] Edmund K. Burke, Mathew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and John R. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In Christine L. Mumford and Lakhmi C. Jain, editors, *Computational Intelligence*, volume 1 of *Intelligent Systems Reference Library*, pages 177–201. Springer Berlin Heidelberg, 2009.

[19] Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. A classification of hyper-heuristic approaches. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 449–468. Springer US, 2010.

[20] Edmund K. Burke, Matthew R. Hyde, Graham Kendall, and John Woodward. Automating the packing heuristic design process with genetic programming. *Evol. Comput.*, 20(1):63–89, March 2012.

[21] E.K. Burke, M.R. Hyde, and G. Kendall. Providing a memory mechanism to enhance the evolutionary design of heuristics. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8, 2010.

[22] E.K. Burke, M.R. Hyde, and G. Kendall. Grammatical evolution of local search heuristics. *Evolutionary Computation, IEEE Transactions on*, 16(3):406–417, 2012.

[23] F. M. Burnet. *The clonal selection theory of acquired immunity*. Cambridge University Press, Cambridge, UK, 1959.

[24] Michael W. Carter, Gilbert Laporte, and Sau Yan Lee. Examination timetabling: Algorithmic strategies and applications. *The Journal of the Operational Research Society*, 47(3):pp. 373–383, 1996.

[25] 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, *Evolutionary Computation in Combinatorial Optimization – EvoCOP2005*, volume 3448 of *LNCS*, pages 23–33, Lausanne, Switzerland, 2005. Springer Verlag.

[26] 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 Berlin Heidelberg, 2008.

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

[28] E. G. Jr. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. pages 46–93, 1997. 241940.

[29] William W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[30] P. Cowling and K. Chakhlevitch. Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In *The 2003 Congress on Evolutionary Computation, CEC 2003*, volume 2, pages 1214–1221, 2003.

[31] Peter I. Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, PATAT '00, pages 176–190. Springer-Verlag, London, UK, 2000.

[32] János Csirik, David S. Johnson, Claire Kenyon, Peter W. Shor, and Richard R. Weber. A self organizing bin packing heuristic. In *Selected papers from the International Workshop on Algorithm Engineering and Experimentation*, ALENEX '99, pages 246–265, London, UK, UK, 1999. Springer-Verlag.

[33] Janos Csirik, David S. Johnson, Claire Kenyon, James B. Orlin, Peter W. Shor, and Richard R. Weber. On the sum-of-squares algorithm for bin packing. *J. ACM*, 53(1):1–65, January 2006.

[34] Dipankar DasGupta. *Artficial Immune Systems and Their Applications*. Springer-Verlag New York, Inc., 1998.

[35] Jörg Denzinger and Matthias Fuchs. High performance atp systems by combining several ai methods. In *Proceedings Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 102–107. Morgan Kaufmann, 1997.

[36] C. Dimopoulos and A. M. S. Zalzala. Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software*, 32(6):489 – 498, 2001.

[37] Philipp A. Djang and Paul R. Finch. Solving one dimensional bin packing problems, 1998.

[38] Edgar A. Duéñez Guzmán and Michael D. Vose. No free lunch and benchmarks. *Evolutionary Computation*, 21(2):293–312, March 2012.

[39] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993.

[40] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[41] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 1186 –1192 vol.2, 1992.

[42] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Working Paper IDSIA-06-99, CRIF Industrial Management and Automation*, 1994.

[43] Emanuel Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evol. Comput.*, 2(2):123–144, 1994.

[44] Emanuel Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley & Sons, Inc., 1998.

[45] Hsiao-Lan Fang, Peter Ross, and Dave Corne. A promising hybrid ga/heuristic approach for open-shop scheduling problems. In A Cohn, editor, *ECAI 94 Proceedings of the 11th European Conference on Artificial Intelligence*, pages 590–594. John Wiley & Sons, Ltd, 1994.

[46] J D Farmer, N H Packard, and A S Perelson. The immune system, adaptation, and machine learning. *Phys. D*, 2(1-3):187–204, 1986.

[47] H Fisher and G L Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In JF Muth and G L Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, New Jersey, 1963.

[48] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 144–151, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[49] Alex A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer, 2002.

[50] Alex S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evol. Comput.*, 16(1):31–61, 2008.

[51] Michael R. Garey and David S. Johnson. *Computers and intractability : a guide to the theory of NP-completeness*. A Series of books in the mathematical sciences. W.H. Freeman, San Francisco, 1979.

[52] Pablo Garrido and Carlos Castro. Stable solving of cvrps using hyperheuristics. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 255–262, New York, NY, USA, 2009. ACM.

[53] Pablo Garrido and Mara Riff. Collaboration between hyperheuristics to solve strip-packing problems. In *Foundations of Fuzzy Logic and Soft Computing*, volume 4529 of *Lecture Notes in Computer Science*, pages 698–707. Springer Berlin / Heidelberg, 2007.

[54] Pablo Garrido and Mara-Cristina Riff. An evolutionary hyperheuristic to solve strip-packing problems. In *Intelligent Data Engineering and Automated Learning - IDEAL 2007*, volume 4881 of *Lecture Notes in Computer Science*, pages 406–415. Springer Berlin / Heidelberg, 2007.

[55] Pablo Garrido and María Riff. Dvrp: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics*, 16: 795–834, 2010.

[56] Christopher D. Geiger, Reha Uzsoy, and Haldun Aytug. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *J. of Scheduling*, 9(1):7–34, 2006.

[57] Ian P. Gent. Heuristic solution of open bin packing problems. *Journal of Heuristics*, 3(4):299–304, 1998.

[58] David Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, (3):493–530, 1989.

[59] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11(1), 2009.

[60] Emma Hart and Peter Ross. A heuristic combination method for solving job-shop scheduling problems. In Agoston Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, pages 845–854. Springer Berlin / Heidelberg, 1998.

[61] Nhu Binh Ho, Joc Cing Tay, and Edmund M. K. Lai. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179(2):316–333, 2007.

[62] Holger Hoos. On the run-time behaviour of stochastic local search algorithms for sat. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 661–666, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[63] Frank Hutter, Holger H. Hoos, Kevin Leyton-brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *JAIR*, 2009.

[64] David Jackson. Single node genetic programming on problems with side effects. In CarlosA.Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 327–336. Springer Berlin Heidelberg, 2012.

[65] David Jackson. A new, node-focused model for genetic programming. In Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado, and Carlos

Cotta, editors, *Genetic Programming*, volume 7244 of *Lecture Notes in Computer Science*, pages 49–60. Springer Berlin Heidelberg, 2012.

[66] Domagoj Jakobovic, Leonardo Jelenkovic, and Leo Budin. Genetic programming heuristics for multiple machine scheduling. In *Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 321–330. Springer Berlin Heidelberg, 2007.

[67] N. K. Jerne. Towards a network theory of the immune system. *Ann Immunol (Paris)*, 125C(1-2):373–89, 1974.

[68] Chuanyi Ji and Sheng Ma. Combinations of weak classifiers. *Neural Networks, IEEE Transactions on*, 8(1):32–42, Jan 1997. ISSN 1045-9227.

[69] Robert Keller and Riccardo Poli. Cost-benefit investigation of a genetic-programming hyperheuristic. In *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 13–24. Springer Berlin / Heidelberg, 2008.

[70] Claire Kenyon. Best-fit bin-packing with random order. In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '96, pages 359–364, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[71] Ashiqur KhudaBukhsh, Lin Xu, Holger Hoos, and Kevin Leyton-Brown. SATenstein: Automatically Building Local Search SAT Solvers From Components. In *International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 2009.

[72] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[73] P. Kromer, J. Platos, and V. Snasel. Practical results of artificial immune systems for combinatorial optimization problems. In *Nature and Biologically Inspired Computing (NaBIC), 2012 Fourth World Congress on*, pages 194–199, 2012.

[74] J. Levine and F. Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *The Journal of the Operational Research Society*, 55(7):705–716, 2004.

[75] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Knowledge Discovery and Data Mining*, pages 80–86, 1998.

[76] Arne Løkketangen and Roland Olsson. Generating meta-heuristic optimization code using adate. *Journal of Heuristics*, 16:911–930, 2010.

[77] Sean Luke and Liviu Panait. A comparison of bloat control methods for genetic programming. *Evol. Comput.*, 14(3):309–344, September 2006.

[78] Javier Marín-Blázquez and Sonia Schulenburg. A hyper-heuristic framework with xcs: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In Tim Kovacs, Xavier Llorà, Keiki Takadama, Pier Lanzi, Wolfgang Stolzmann, and Stewart Wilson, editors, *Learning Classifier Systems*, volume 4399 of *Lecture Notes in Computer Science*, pages 193–218. Springer Berlin / Heidelberg, 2007.

[79] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[80] Silvano Martello and Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28(1):59–70, 1990.

[81] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, pages 321–326. AAAI Press, 1997.

[82] T.E. Morton and D.W. Pentico. *Heuristic scheduling systems: with applications to production systems and project management*. Wiley series in engineering and technology management. Wiley, 1993.

[83] Alexander Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In Mauricio G. C. Resende and Jorge Pinho de Sousa, editors, *Metaheuristics: computer decision-making*, chapter 9, pages 523–544. Kluwer Academic Publishers, Norwell, MA, USA, 2004. ISBN 1-4020-7653-3.

[84] Igor Norenkov and Erik D. Goodman. Solving scheduling problems via evolutionary methods for rule sequence optimization. In *2nd World Conference on soft Computing, WSC2*, 1997.

[85] G. Ochoa, M. Hyde, T. Curtois, J.A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A.J. Parkes, S. Petrovic, and E.K. Burke. HyFlex: A Benchmark Framework for Cross-domain Heuristic Search. 2012.

[86] Gabriela Ochoa. A bibliography of hyper-heuristics and related approaches. Retreived July 2011 from http://www.cs.nott.ac.uk/ gxo/hhbibliography.html.

[87] Roland Olsson. Inductive functional programming using incremental program transformation. *Artif. Intell.*, 74:55–81, March 1995.

[88] Mihai Oltean and D. Dumitrescu. Evolving tsp heuristics using multi expression programming. In Marian Bubak, GeertDick Albada, PeterM.A. Sloot, and Jack Dongarra, editors, *Computational Science - ICCS 2004*, volume 3037 of *Lecture Notes in Computer Science*, pages 670–673. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22115-9.

[89] EURO Special Interest Group on Cutting and Packing (EPICUP). Data sets 1d. Retreived August 2011 from `http://paginas.fe.up.pt/~esicup/tiki-list_file_gallery.php`.

[90] M. O'Neill and C. Ryan. Grammatical evolution. *Evolutionary Computation, IEEE Transactions on*, 5(4):349–358, 2001.

[91] Ender Özcan and Andrew J. Parkes. Policy matrix evolution for generation of heuristics. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 2011–2018, New York, NY, USA, 2011. ACM.

[92] Ender Özcan, Burak Bilgin, and Emin Korkmaz. Hill climbers and mutational heuristics in hyperheuristics. In *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 202–211. Springer Berlin / Heidelberg, 2006.

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

[94] N. Pillay. Evolving hyper-heuristics for the uncapacitated examination timetabling problem. *Journal of the Operational Research Society*, 63:47–58, 2012.

[95] R. Poli, J. Woodward, and E. K. Burke. A histogram-matching approach to the evolution of bin-packing strategies. In *IEEE Congress on Evolutionary Computation, 2007. (CEC 2007)*, pages 3500–3507. Singapore, 2007.

[96] Mitchell A. Potter and Kenneth A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evol. Comput.*, 8:1–29, 2000.

[97] Rong Qu and Edmund K Burke. Hybridisations within a graph based hyperheuristic framework for university timetabling problems. *Journal of the Operational Research Society*, 60:1273–1285, 2009.

[98] Rong Qu, Edmund K. Burke, and Barry McCollum. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research*, 198(2):392–404, 2009.

[99] John R Rice. The algorithm selection problem. In Morris Rubinoff and Marshall C. Yovits, editors, *Advances in Computers*, volume 15, pages 65 – 118. Elsevier, 1976.

[100] Peter Ross. Hyper-heuristics. In Graham Burke, Edmund K. Kendall, editor, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 529–556. Springer-Verlag, 2005.

[101] Peter Ross, Sonia Schulenburg, Javier G. Marín-Blázquez, and Emma Hart. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 942–948, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[102] Peter Ross, Javier Marín-Blázquez, Sonia Schulenburg, and Emma Hart. Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics. In Erick Cantú-Paz, James Foster, Kalyanmoy Deb, Lawrence Davis, Rajkumar Roy, Una-May OReilly, Hans-Georg Beyer, Russell Standish, Graham Kendall, Stewart Wilson, Mark Harman, Joachim Wegener, Dipankar Dasgupta, Mitch Potter, Alan Schultz, Kathryn Dowsland, Natasha Jonoska, and Julian Miller, editors, *Genetic and Evolutionary Computation GECCO 2003*, volume 2724 of *Lecture Notes in Computer Science*, pages 215–215. Springer Berlin / Heidelberg, 2003.

[103] A Scholl and R Klein. Bin packing. Retrieved March 2011, from http://www.wiwi.uni-jena.de/Entscheidung/binpp/.

[104] Armin Scholl, Robert Klein, and Christian Jürgens. Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Comput. Oper. Res.*, 24(7):627–645, 1997.

[105] P. Schwerin and G. Wäscher. The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp. *International Transactions in Operational Research*, 4(5-6):377–389, 1997.

[106] Bart Selman and Henry Kautz. Domain-independent extensions to gsat: solving large structured satisfiability problems. In *Proceedings of the 13th international joint conference on Artifical intelligence - Volume 1*, pages 290–295, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[107] Bart Selman, Hector Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446. AAAI Press, 1992.

[108] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the twelfth national conference on Artificial intelligence*, pages 337–343, Seattle, Washington, United States, 1994. American Association for Artificial Intelligence.

[109] K. Sim. Ksats-hh: A simulated annealing hyper-heuristic with reinforcement learning and tabu-search. Entry in the Cross-domain Heuristic Search Challenge available from `http://www.asap.cs.nott.ac.uk/external/chesc2011/index.html`, June 2011.

[110] K. Sim. Asynchronous idiotypic network simulator. In Emma Hart, Jon Timmis, Paul Mitchell, Takadash Nakamo, Foad Dabiri, Ozgur Akan, Paolo Bellavista, Jiannong Cao, Falko Dressler, Domenico Ferrari, Mario Gerla, Hisashi Kobayashi, Sergio Palazzo, Sartaj Sahni, Xuemin (Sherman) Shen, Mircea Stan, Jia Xiaohua, Albert Zomaya, and Geoffrey Coulson, editors, *Bio-Inspired Models of Networks, Information, and Computing Systems*, volume 103 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 248–251. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-32711-7.

[111] K Sim. One dimensional bin packing problems. Available from `http://www.soc.napier.ac.uk/~cs378/bpp/`, 2013.

[112] K. Sim, E. Hart, and B. Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation Journal*, In Press, January 2014.

[113] Kevin Sim and Emma Hart. Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In Christian Blum, editor, *GECCO '13: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, New York, NY, USA, 2013. ACM.

[114] Kevin Sim and Emma Hart. An improved immune inspired hyper-heuristic for combinatorial optimisation problems. In *GECCO '14: Proceeding of the six-*

teenth annual conference on Genetic and evolutionary computation conference, 2014.

[115] Kevin Sim, Emma Hart, and Ben Paechter. A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In CarlosA.Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 348–357. Springer Berlin Heidelberg, 2012.

[116] Kevin Sim, Emma Hart, and Ben Paechter. Learning to solve bin packing problems with an immune inspired hyper-heuristic. In Giuseppe Nicosia Stefano Nolfi Pietro Lió, Orazio Miglino and Mario Pavone, editors, *Advances in Artificial Life, ECAL 2013: Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems*, pages 856–863. MIT Press, 2013.

[117] K. A. Smith-Miles. Towards insightful algorithm selection for optimisation using meta-learning concepts. pages 4118–4124, 2008.

[118] Kate Smith-Miles, Brendan Wreford, Leo Lopes, and Nur Insani. Predicting metaheuristic performance on graph coloring problems using data mining. In El-Ghazali Talbi, editor, *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 417–432. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-30670-9.

[119] Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1):1–25, 2008. 1456656.

[120] Robert H. Storer, S. David Wu, and Renzo Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):pp. 1495–1509, 1992.

[121] Robert H. Storer, S. David Wu, and Renzo Vaccari. Problem and heuristic space

search strategies for job shop scheduling. *Informs Journal on Computing*, 7(4): 453–467, 1995.

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

[123] Joc Cing Tay and Nhu Binh Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453–473, 2008.

[124] H. Terashima-Marín, E. J. Flores-Álvarez, and P. Ross. Hyper-heuristics and classifier systems for solving 2d-regular cutting stock problems. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 637–643, New York, NY, USA, 2005. ACM.

[125] H. Terashima-Marín, C. J. Farías Zárate, P. Ross, and M. Valenzuela-Rendón. A ga-based method to produce generalized hyper-heuristics for the 2d-regular cutting stock problem. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 591–598. ACM, New York, NY, USA, 2006.

[126] H. Terashima-Marin, C. J. Farias Zarate, P. Ross, and M. Valenzuela-Rendon. Comparing two models to generate hyper-heuristics for the 2d-regular bin-packing problem. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 2182–2189. ACM, New York, NY, USA, 2007.

[127] H. Terashima-Marín, J. C. Ortiz-Bayliss, P. Ross, and M. Valenzuela-Rendón. Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 571–578, New York, NY, USA, 2008. ACM.

[128] H. Terashima-Marín, P. Ross, C. Farías-Zárate, E. López-Camacho, and M. Valenzuela-Rendón. Generalized hyper-heuristics for solving 2d regular

and irregular packing problems. *Annals of Operations Research*, 179:369–392, 2010.

[129] Hugo Terashima-Marín and Peter Ross. Evolution of constraint satisfaction strategies in examination timetabling. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99*, pages 635–642. Morgan Kaufmann, 1999.

[130] F. Thabtah, P. Cowling, and Y. Peng. Mcar: multi-class classification based on association rule. In *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on*, page 33, 2005.

[131] F.A. Thabtah, P. Cowling, and Yonghong Peng. Mmac: a new multi-class, multi-label associative classification approach. In *Data Mining, 2004. ICDM '04. Fourth IEEE International Conference on*, pages 217 – 224, 2004.

[132] Fadi Thabtah and Peter Cowling. Mining the data from a hyperheuristic approach using associative classification. *Expert Systems with Applications*, 34 (2):1093–1101, 2008.

[133] Dorndorf Ulrich and Pesch Erwin. Evolution based learning in a job shop scheduling environment. *Comput. Oper. Res.*, 22(1):25–40, 1995. 197887.

[134] Stewart W Wilson. Classifier fitness based on accuracy. *Evol. Comput.*, 3(2): 149–175, 1995.

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