Hyper-heuristic Decision Tree Induction

Alan Vella

Submitted for the degree of Doctor of Philosophy

Heriot-Watt University

The School of Mathematical and Computer Sciences

March 2012

Abstract

A hyper-heuristic is any algorithm that searches or operates in the space of heuristics as opposed to the space of solutions. Hyper-heuristics are increasingly used in function and combinatorial optimization. Rather than attempt to solve a problem using a fixed heuristic, a hyper-heuristic approach attempts to find a combination of heuristics that solve a problem (and in turn may be directly suitable for a class of problem instances). Hyper-heuristics have been little explored in data mining. This work presents novel hyper-heuristic approaches to data mining, by searching a space of attribute selection criteria for decision tree building algorithm. The search is conducted by a genetic algorithm. The result of the hyper-heuristic search in this case is a strategy for selecting attributes while building decision trees.

Most hyper-heuristics work by trying to adapt the heuristic to the state of the problem being solved. Our hyper-heuristic is no different. It employs a strategy for adapting the heuristic used to build decision tree nodes according to some set of features of the training set it is working on. We introduce, explore and evaluate five different ways in which this problem state can be represented for a hyper-heuristic that operates within a decision-tree building algorithm. In each case, the hyper-heuristic is guided by a rule set that tries to map features of the data set to be split by the decision tree building algorithm to a heuristic to be used for splitting the same data set. We also explore and evaluate three different sets of low-level heuristics that could be employed by such a hyper-heuristic.

This work also makes a distinction between *specialist* hyper-heuristics and *generalist* hyper-heuristics. The main difference between these two hyper-heuristcs is the number of training sets used by the hyper-heuristic genetic algorithm. Specialist hyper-heuristics are created using a single data set from a particular domain for evolving the hyper-heurisic rule set. Such algorithms are expected to outperform standard algorithms on the kind of data set used by the hyper-heuristic genetic algorithm. Generalist hyper-heuristics are trained on multiple data sets from different domains and are expected to deliver a robust and competitive performance over these data sets when compared to standard algorithms.

We evaluate both approaches for each kind of hyper-heuristic presented in this thesis. We use both real data sets as well as synthetic data sets. Our results suggest that none of the hyper-heuristics presented in this work are suited for specialization – in most cases, the hyper-heuristic's performance on the data set it was specialized for was not significantly better than that of the best performing standard algorithm. On the other hand, the generalist hyper-heuristics delivered results that were very competitive to the best standard methods. In some cases we even achieved a significantly better overall performance than all of the standard methods.

Acknowledgements

My supervisor, Prof. David W. Corne, for his invaluable help and guidance.

Chris Murphy (Motorola), for the helpful discussions.

My wife, Véronique, for her infinite support and encourgement.

ACADEMIC REGISTRY
Research Thesis Submission

Name:	Alan Vella		
School/PGI:	School of Mathematical and Computer Sciences		
Version:	Final	Degree Sought	PhD in Computer Science

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1) the thesis embodies the results of my own work and has been composed by myself
2) where appropriate, I have made acknowledgement of the work of others and have made reference to work carried out in collaboration with other persons
3) the thesis is the correct version of the thesis for submission and is the same version as any electronic versions submitted*.
4) my thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
5) I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.

* *Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.*

Signature of Candidate:		Date:	

Submission

Submitted By *(name in capitals):*	
Signature of Individual Submitting:	
Date Submitted:	

For Completion in the Student Service Centre (SSC)

Received in the SSC by			
Method of Submission			
E-thesis Submitted			
Signature:		Date:	

Table of Contents

List of Tables

List of Figures

List of Acronyms

DTB	Decision Tree Builder
DTBA	Decision Tree Building Algorithm
CHI	Chi-Squre
GINI	Gini Index
GR	Gain Ratio
HH	Hyper-heuristic
HH-5	Hyper-heuristic Using 5 Heuristics
HH-12	Hyper-heuristic Using 12 Heuristics
HHTRS	Hyper-heuristic Training Set
HHTRE	Hyper-heuristic Test Set
HHx	Hyper-heuristic Trained on x Data Sets
IG	Informatin Gain
JM	J-Measure
MCE	Maximum Conditional Entropy
MDL	Minimum Description Length
MGINI	Modified Gini Index
NSDTB	Not Significantly Different To The Best
RLV	Relevance
RLF	Relief
SDS	Synthetic Data Set
SGAIN	Symmetric Gain
SGINI	Symmetric Gini
WOE	Weight of Evidence

Chapter 1

Introduction

1.1 Data Mining

Data mining is the extraction of useful knowledge from data (Freitas, 2002). One very common data mining task is classification. Classification involves predicting the class of previously unseen objects using a number of attributes that describe these objects. This could be trying to guess whether a patient has some form of cancer or not from his or her symptoms and physical attributes; or trying to predict the likelihood of a customer being interested in a certain product given a history of his or her past purchases. This is made possible through the use of a classification system that is built by generalising from available training data – data that contains a set of instances together with their class.

There are many different ways of representing a classification model: classification rules, decision trees, Bayesian networks, neural networks and support vectors produced by support vector machines are examples of such representations. Each representation comes with its own set of advantages and disadvantages. When trying to choose a suitable representation for a particular problem, one usually has to consider the pros and cons of each type of classifier against the requirements of the solution that is needed. For example, a neural network might be very accurate in predicting the class of an unseen object but it is no use to someone who wants to understand the mechanism used by the classifier to separate instances into classes. Classification rules might be a better option is such a case since such rules are very easy to read and understand.

1.2 Motivation for Hyper-heuristics

Classification is used in a wide variety of problem domains. There are numerous algorithms that can be used to build each of the representations a classifier can take. It has been shown that there is no such thing as a single algorithm that works well for all problem domains (Lim et al, 2000). This creates the challenge of deciding which is the most suitable algorithm to use for building the classifier from the available training data. For example, there are many different decision tree building algorithms that can be used to build a decision tree. Different algorithms use different heuristics for building a decision tree and there is no clear way of matching the right algorithm to the data that needs to be mined. Furthermore, it rarely happens that an algorithm is used as is. There is often a customisation process that involves fine tuning and tailoring the data mining algorithm to the nature of the training data set at hand.

The problem of choosing the right algorithm for a given problem also exists outside the field of data mining. For example, Soubeiga (2003) explains the difficulty of re-using algorithms for building schedules over different problem cases. An algorithm that works well for one case of a scheduling problem is bound to perform very badly on another case of a completely different scheduling problem. Burke et al (2003b) highlight the same kind of issues when working with algorithms that build timetables and rosters while Ross et al (2003) give examples that demonstrate similar challenges in the field of bin-packing problems.

One way of overcoming this problem is through the use of *hyper-heuristics* (Cowling et al, 2000; Özcan et al, 2008). A hyper-heuristic is a strategy for adapting the heuristic used to build a solution according to the problem at hand. This is possible because hyper-heuristics, unlike standard heuristics and meta-heuristics, operate in the space of heuristics as opposed to the space of solutions. In the past decade or so, research in this area has produced many different types of hyper-heuristics. A description that probably holds true for most (if not all) of these hyper-heuristics is one that defines a hyper-heuristic as a program that takes a problem state and/or a solution state as input and returns the heuristic to be applied to that problem as output (see Figure 1.1).

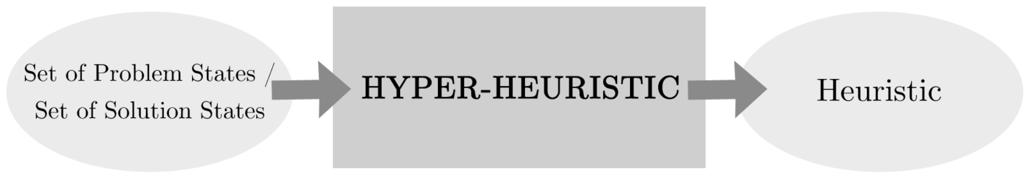

Figure 1.1: Concept of Hyper-heuristics

Hyper-heuristics have been successfully applied to many kinds of scheduling problems (Fang et al, 1993; 1994; Cowling et al, 2000; 2000a; Kendall et al 2002a; Cowling and Chakhlevitch, 2003; Chakhlevitch and Cowling, 2005. Burke et al, 2005b), timetabling problems (Burke and Newall, 2002; Burke et al, 2003b; 2005a; 2007a; 2007b; Ross et al, 2004; Ross and Marín-Blázquez, 2005; Chen et al, 2007; Qu and Burke, 2008), cutting stock problems (Terashima-Marín et al, 2005; 2006), bin-packing problems (Ross et al, 2002;

2003; Burke et al, 2006a; Marín-Blázquez and Schulenburg, 2006; 2007) as well as a number of other combinatorial and constraint satisfaction problems (Schmiedle et al, 2002; Kendall and Mohammad, 2004a; Oltean and Dumitrescu, 2004; Keller and Poli 2007a; 2007b; Kumar et al, 2008).

1.3 Hyper-heuristics for Decision Tree Building Algorithms

Very little work has been done in the way of applying hyper-heuristics to create adaptive data mining algorithms. To the best of our knowledge, the only work that does this is Pappa and Freitas (2006). The idea was first presented in Pappa and Freitas (2004). This work uses grammar-based genetic programming to automatically construct rule induction algorithms. Though the term hyper-heuristics is never explicitly mentioned by the authors of this work, their method involves conducting a search in the space of heuristics as opposed to the space of solutions. To be precise, the search is conducted via genetic programming in a space made up of the basic building blocks of rule induction algorithms.

This thesis presents an alternative hyper-heuristic framework for constructing a different breed of data mining algorithms: decision tree building algorithms. Decision trees are tree-like graphs made up of nodes, branches and leaves where the nodes contain attribute names, the branches contain attribute values and the leaves contain target class values (see Figure 1.2). A decision tree can be used to classify objects by traversing it from the top down. At each step of the way, the branch taken is always the one whose value matches the value of the object's attributes. The traversal process stops at one of the bottom leaves. The class specified by the leaf is the class assigned to the object. Decision trees are widely used in the data mining community as they are easy to read and understand as well as fast to traverse.

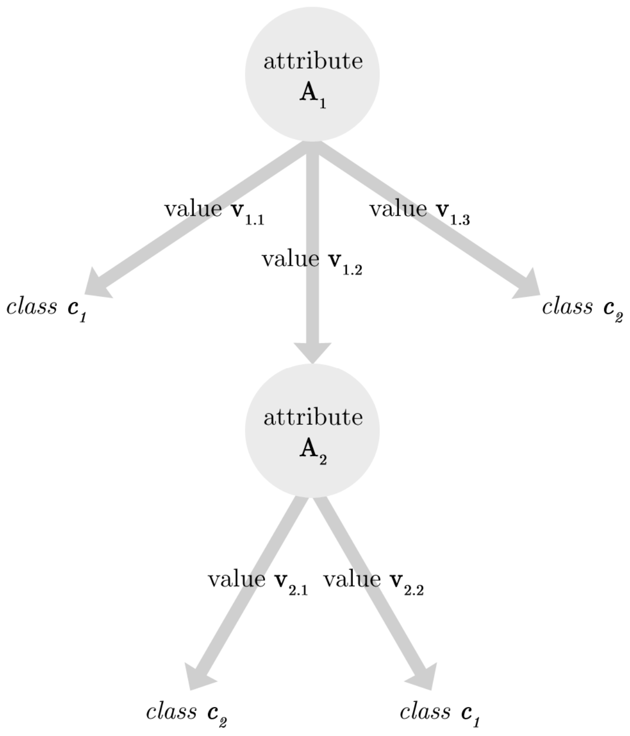

Figure 1.2: A Generic Decision Tree

A typical decision tree building algorithm builds a tree step by step by deciding, at each step, how to develop the next node in the tree using the training data set. The topmost node is the first to be created after which the rest of the tree is grown in a recursive manner. An example of such an algorithm is the popular ID3 (Quinlan, 1986). The vast majority of algorithms for building decision trees use this strategy. One of the procedures that usually varies from one algorithm to another is the heuristic used to choose which attribute should be placed at each node in the tree. ID3 uses a heuristic based on information gain to create nodes in a decision tree. The heuristic used to create tree nodes is crucial to the performance of the resulting decision tree, so much so that decision tree building algorithms are often defined, to some extent, by the heuristic they employ to create nodes (Buntine and Niblett, 1992; Liu and White, 1994).

One thing common to all decision tree building algorithms is that this heuristic is static throughout the whole tree building process. This brings us back to the problem mentioned earlier. There is no heuristic that is universally better at building decision trees. Different heuristics are suited for different data sets. Furthermore, there is no clear way of choosing which heuristic to use according to the data set at hand.

This work addresses this problem by proposing a hyper-heuristic decision tree induction algorithm. Our hyper-heuristic automatically chooses which heuristic to use according to the data set at hand. We do this by augmenting the standard decision tree building technique described earlier with a hyper-heuristic that, at each step of the way, is input with the current problem state and returns the heuristic to be used to handle that problem state. The problem state is the training data set while the output is the heuristic that chooses which attribute to use at the next tree node to be created. The hyper-heuristic manages this through a rule set: a number of if-then rules that map certain features of the data set to a heuristic (see Figure 1.3).

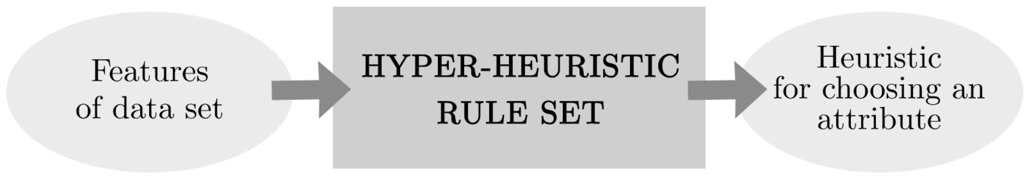

Figure 1.3: Hyper-heuristic for Decision Tree Induction

One can call this rule set the “brain” of the hyper-heuristic since it transforms our decision tree building algorithm into a hyper-heuristic capable of adapting the heuristic used for attribute selection according to the problem at hand.

We present various ways in which such a rule set can be represented, mostly differing in the way the problem state is characterized (i.e. the data set features considered by the rule set). We explore two possible uses for each of these variants by defining two types of hyper-heuristics for decision tree building algorithms: *specialized hyper-heuristics* that are tailored for a data set from a particular problem domain and *generalized hyper-heuristics* that are expected to run reasonably well over a number of data sets from different problem domains. In all cases, we use a hyper-heuristic genetic algorithm for discovering a good set of rules.

All of these hyper-heuristics are tested on a number of real data sets picked from a wide variety of domains. We use data sets of different sizes with underlying classification models of varying complexity. The results obtained are compared to those achieved by a number of standard, non-adaptive decision tree building algorithms. We identify situations in which using such hyper-heuristics can yield decision trees of a higher predictive accuracy than the ones created by the standard methods. We also run experiments on synthetic data sets to try and understand how such hyper-heuristics are capable of adapting the heuristic used according to the problem state.

1.4 Contributions

This thesis offers contributions in the area of hyper-heuristics and data mining. The primary contribution of this thesis is the first set of approaches to develop decision-tree based data mining algorithms using hyper-heuristics. In fact, this represents one of the first few attempts at developing data mining algorithms in general using hyper-heuristics. A specific breakdown of the main contributions is as follows:

1. A key factor for any hyper-heuristic is how to represent the problem state. We introduce, explore and evaluate five different ways how the problem state can be represented:
    a. Using the number of instances in the partition to be split (sections 3.4 and 3.5).
    b. Using the number of attributes left in the partition to be split (section 4.1).
    c. Using the value count of the attributes left in the partition to be split (section 4.2).
    d. Using the entropy of the attributes left in the partition to be split (section 4.3).
    e. Using the maximum conditional entropy of the class attribute in the partition to be split (section 4.4).

2. Another key factor for any hyper-heuristic is the pool of low-level heuristics at its disposal. We explore and evaluate three different pools of heuristics for finding splitting attributes:
    a. Using a single fixed heuristic for sorting the candidate splitting attributes while adapting the ranking of the chosen attribute (section 3.4).
    b. Using a set of two heuristics for sorting the candidate splitting attributes as well as adapting the ranking of the chosen attribute (section 3.5).
    c. Using a bigger set of five or twelve heuristics for sorting the candidate splitting attributes while always choosing the first-ranked attribute from the sorted list of attributes (sections 4.1, 4.2, 4,3 and 4.4).

3. We make a distinction between specialist hyper-heuristics (trained on a single data set from a particular domain) and generalist hyper-

heurisitics (trained on a number of data sets from different domains) and we evaluate both approaches for each of the hyper-heuristics presented in this work.

4. In chapter 5 we present experimental evidence using synthetic data sets that tries to shed some light on when and why our hyper-heuristics work. These experiments suggest that the number of data sets used to train a hyper-heuristic for decision-tree building algorithms such as the ones presented in this work is related to the performance of the resultant decision trees (section 5.1). They also suggest that having a pool of training data sets with a variety of class distributions can help evolve more robust hyper-heuristics (section 5.2).

The PhD research described in this thesis has produced the following publication:

Vella A., Corne D. and Murphy C. (2009) *Hyper-heuristic Decision Tree Induction.* World Congress on Nature & Biologically Inspired Computing, 2009, pp. 409-414.

1.5 Thesis Overview

The rest of this thesis is structured as follows. Chapter 2 provides the necessary background to this thesis in the form of a literature review of hyper-heuristics and decision-tree building algorithms. In chapter 3 we discuss certain shortcomings of the current non-adaptive methods for building decision trees, thus explaining our motivation for using hyper-heuristics. We present the first version of our hyper-heuristic together with experimental results that compare it to standard methods on real data sets. Chapter 4 presents four more variants of our hyper-heuristic. Each of these variants is again compared to standard methods on real data sets. In chapter 5 we try to understand how and when our hyper-heuristics perform well by running experiments on synthetic data sets. Chapter 6 lists our conclusions as well as possible future work.

Chapter 2

Literature Review

2.1 Hyper-heuristics

2.1.1 Origins and Early Approaches

The origin of the notion behind hyper-heuristics can be traced back to the 1960s (Fisher and Thompson, 1961; Crowston et al, 1963). This work investigated combinations of basic rules for job-shop scheduling. Using probabilistic learning they showed that combining these scheduling rules gave better results than applying any one of them separately. This notion was not revisited until 1985 (Smith, 1985) when Smith developed a hybrid genetic algorithm for solving the bin packing problem. This genetic algorithm used an indirect encoding approach in which a chromosome represented rules for producing the solution as opposed to the solution itself. In this work, a base heuristic was used to pack items of various sizes into bins while a genetic algorithm was employed to find the best ordering for these items. From 1985 until the late 1990s a series of works in the field of hyper-heuristics were carried out although the term "hyper-heuristic" was not introduced until 2000

(Cowling et al, 2000). What unites these works is that they all employ methods that conduct a search or operate in the space of heuristics (or parameters for heuristics) to solve a set of instances of a problem. This is in contrast to other search methods that work in the space of direct solutions.

Syswerda (1991) and Fang et al (1993) both applied a genetic algorithm that used an indirect encoding to scheduling where a heuristic was used to fit an ordered list of tasks into a developing schedule. The genetic algorithm was used to find the optimal ordering for these tasks. Kelly and Davis (1991) applied a similar hybrid genetic algorithm approach to k-nearest neighbour to aid data classification while Ling (1992) used this technique for solving timetabling problems. Shing and Parker (1993) used a hybrid genetic algorithm to optimise parameters for a set of heuristics. Another notable study was that of Gratsch et al (1993; 1996), which used hill-climbing in a space of control strategies to find good algorithms for controlling satellite communication schedules.

Storer et al. (1992; 1995) identified the need to carry out a search in the heuristic space so as to generate a combination of problem-specific heuristics. They used a hill-climbing technique to search within this neighbourhood but they also suggested that other search techniques could be used such as genetic algorithms, tabu search or simulated annealing. The open-shop scheduling problem was revisited by Fang et al in 1994. This time they used a genetic algorithm to evolve a sequence of heuristics to be applied to the problem. In this work the authors comment on the advantages of using a genetic algorithm hybridized with a heuristic search where each chromosome represents an abstraction of a solution. Such hybridization avoids the need to represent a complete solution as an individual in the population of a genetic algorithm, thus facilitating the search performed by the genetic algorithm.

In Langdon (1995) a hybrid genetic algorithm was used to schedule the maintenance of electrical power transmission networks. This problem was revisited in Langdon (1996) in which genetic programming was used to evolve a schedule for the same purpose. Dorndorf and Pesch (1995) used evolutionary algorithms to evolve local decision rule sequences for production scheduling. Hybrid genetic algorithm approaches were used to solve a graph colouring problem in Fleurent and Ferland (1996), a bin-packing problem in Reeves (1996), a line-balancing problem in Schaffer and Eshelman (1996) and timetabling problems in Corne and Ogden (1997) and Terashima- Marín et al (1999). In Terashima-Marin et al (1999), an individual represented a series of instructions and parameters for guiding a search algorithm that builds a timetable. The instructions allowed the algorithm to switch its search strategy when a certain problem condition is met.

Zhang and Dietterich (1995; 1996) developed novel job-shop scheduling heuristics within a reinforcement learning framework while Minton (1996) presented a system that works by modifying elements of a template for a generic algorithm. This system was used to generate reusable heuristics for constraint satisfaction problems. Norenkov and Goodman (1997) proposed a system that conducts a search in the space of heuristics using evolutionary algorithms to solve multistage flow-shop scheduling problems. Hart and Ross (1998) evolved heuristic combinations for dynamic job-shop scheduling problems using a genetic algorithm. This concept was later applied by Hart et al (1998) to a real-world scheduling problem. The problem involved scheduling the collection and delivery of chickens from farms in different locations to multiple processing factories. This time two genetic algorithms were used, one to evolve heuristics for assigning orders and the other to evolve heuristics for scheduling the arrival of deliveries.

A novel search framework called "squeaky wheel optimization" was proposed by Joslin and Clements (1999) that works by iterating three stages named *construct*, *analyze* and *prioritize*. In the *construct* stage, a heuristic is used to construct a complete solution using some priorities related to features of the problem being solved. In the *analyze* stage, the resultant solution is analyzed and "trouble spots" are identified. Trouble spots are elements in the solution that if improved are likely to increase the objective function score. In the *prioritize* stage the priorities are redefined according to the identified "trouble spots". The idea is to operate the search in the space of priorities as opposed to the space of solutions. In the same year, Voudouris and Tsang (1999) proposed a hillclimbing algorithm that is re-run with a modified target function each time it gets stuck in a local minimum. The target function is changed through the use of problem-dependent features which carry costs and penalties.

2.1.2 Reinforcement Learning Hyper-heuristics

The literature contains several hyper-heuristics that use reinforcement learning. Such hyper-heuristics would typically utilize a pool of low-level heuristics and some technique for choosing which heuristic to apply to the solution at the different stages of the problem-solving process. The decision would be based on some sort of feedback relating to the heuristics' past performance when applied to the problem. One such hyper-heuristic is the choice function hyper-heuristic which was first proposed by Cowling et al (2000). This hyper-heuristic iteratively modifies a candidate solution using a set of heuristics. The choice function chooses which heuristic to apply at each iteration by taking into consideration the past performance of each heuristic, the heuristic applied just before as well as the last time each heuristic has been called. This system was successfully applied to a hard scheduling problem in (Cowling et al, 2002a) and to a real-world nurse scheduling problem in (Kendall et al, 2002a). Further experiments in (Kendall et al, 2002b) confirmed that the choice function does adapt and choose intelligently which low-level heuristics to call in the order that best suits the search space and the problem being solved.

Narayek (2001) proposed a hillclimbing algorithm that has a set of heuristics at its disposal where each heuristic is assigned a weight value. At each iteration, the system always applies the heuristic with the largest weight value. The weight values are modified (adapted) according to the heuristics' past performance during the search process. A similar heuristic-ranking system was presented by Burke et al (2003b) who applied this concept to a timetabling problem. This hyper-heuristic also made use of a tabu list that temporarily omits heuristics from being chosen if they do not improve the solution. Burke and Newall (2002) presented a hyper-heuristic framework for solving exam timetabling problems that iteratively adapts the ranking of exams left to schedule for a heuristic that inserts these exams into a developing timetable. A weighting system is again used to change the ranking of exams left to schedule as the algorithm progresses. A heuristic-ranking hyper-heuristic is described in (Pisinger and Ropke, 2007) for solving vehicle routing problems. This hyper-heuristic uses roulette wheel selection to choose which heuristic to apply at each iteration. The better the past performance of a heuristic, the higher its weight value and the higher the probability of it being chosen by the hyper-heuristic at the next iteration.

Dowsland et al (2007) proposed a novel hyper-heuristic embedded within a simulated annealing framework. The system was used to determine shipper sizes for storage and transportation – a complex problem with a massive search space. The hyper-heuristic employs a set of heuristics and the same heuristic-ranking technique is used where the heuristic with the highest weight is always chosen. A tabu list is also used for temporarily omitting heuristics that do not improve the solution. At each iteration, an acceptance mechanism is used to decide whether a new solution is accepted or not after the chosen heuristic is applied. The acceptance method for a move in the solution search space is based on simulated annealing but the temperature can go both up and down during the course of the hyper-heuristic run. A similar simulated annealing based hyper-heuristic was presented in the same year by Bai et al

(Bai et al, 2007a). This time, the hyper-heuristic employs a short-term memory so that only the recent performance of the heuristic is reflected in its weight value. The authors' reason for using such a technique is that the search for a good solution proceeds in a dynamic environment as the region of the search space being explored changes and the information gathered during the initial stages may not be useful later in the search. The exam timetabling problem was revisited by Burke et al (2008b). This work presented a hyper-heuristic that combined reinforcement learning (through heuristic-ranking) with the Great Deluge acceptance mechanism (Kendall and Mohamad, 2004a).

2.1.3 Tabu-Search Hyper-heuristics

Burke et al (2003b) proposed a tabu-search hyper-heuristic for solving timetabling problems. The system builds a timetable iteratively by applying a heuristic from a set at each iteration. The heuristic to be applied is chosen on the strength of its past performance. If, once applied, the heuristic does not improve the solution, it is inserted in a tabu list for some time so as to temporarily prevent it from being chosen. This approach was later integrated within a simulated annealing framework by Dowsland et al in 2007 for the purpose of searching for good combinations of low-level heuristics to determine shipper sizes in transportation problems. The timetabling problem was also tackled by Kendall and Hussin (2005) who used a hyper-heuristic to build timetables in the same iterative manner as described in Burke et al (2003). In this work, all heuristics are tried out on the solution at each iteration but only the heuristic that yields the best improvement is actually applied to the solution before moving on to the next iteration. After the best heuristic is applied to the solution, it is inserted in a tabu list so as to exclude it from being chosen by the hyper-heuristic for some time. The authors argue that such a system allows for a balance between intensification (best heuristic is always chosen) and diversification (heuristic is excluded for some time after it is applied) in terms of the search process.

In Cowling and Chakhlevitch (2003) and Chakhlevitch and Cowling (2005), the performances of various hyper-heuristics that make use of a collection of low-level heuristics were investigated by testing them on personnel scheduling problems. Four hyper-heuristics that use some kind of tabu list were compared to various other hyper-heuristics including four Peckish hyper-heuristics (Corne and Ross, 1995). The Peckish algorithm works by always applying the heuristic that gives the biggest improvement in terms of the solution objective function. If none of the heuristics manage to improve the solution, a random one is chosen and applied. The results of their experiments suggested that using smaller sets of low-level heuristics was not so effective and that the performance of a hyper-heuristic is determined to a great extent by the quality of the low-level heuristics used. The same conclusion was arrived at by Özcan et al (2008) who compared the performance of genetic algorithms and memetic algorithms to a number of hyper-heuristic methods that use various combinations of different heuristic selection and acceptance methods.

In Burke et al (2005a; 2007a) and Qu and Burke (2008), tabu search based hyper-heuristic methods were used to find heuristic lists to solve timetabling problems. The search space of the tabu search consisted of all possible permutations of low-level graph colouring heuristics that construct timetables. In Burke et al (2007a), the authors highlight one advantage of conducting a search in the heuristic space by saying that moving in the heuristic space can result in jumps within the solution space that might not be always possible if the search is conducted in the solution space. They do however mention that a search in the heuristic space might make some solutions unreachable. The authors also conclude that the larger the number of low-level heuristics available to the hyper-heuristic, the better it may perform provided a reasonable search time is given.

2.1.4 Hyper-heuristics that use Problem State Representation

In Petrovic and Qu (2002) and Burke et al (2002), a system was proposed that matches the features of some instance of a timetabling problem with case problems stored in a case base. Each entry in the case base would contain the heuristic most suited for solving the case problem. Ross et al. (2002) successfully used a Michigan-style (Freitas, 2002) learning classifier system called XCS (Wilson, 1995) to evolve a set of rules that dictate which heuristic to use when some particular problem state is encountered while solving a bin-packing problem. The problem state is represented by the number and size of items left to pack. This way, the hyper-heuristic manages which heuristics are used as well as when they are used throughout the problem-solving process. Marín-Blázquez and Schulenburg (2006; 2007) extended this system by introducing a multi-step reward system. The produced algorithms generalized well on both training and unseen data. The authors stressed the importance of the choice of representation for the problem state in the hyper-heuristic rules.

In Ross et al (2003), a messy genetic algorithm was used to evolve the same kind of hyper-heuristic rules described in Ross et al (2002). As in their previous work, each chromosome represents a set of rules that match heuristics to problem states. In the first generation of the genetic algorithm, each chromosome was applied to 5 bin-packing problems chosen randomly from a set of training problems. In the subsequent generations, each chromosome was assigned one random problem so as to calculate its fitness. The fitness value was taken to be the difference between the results obtained by the hyper-heuristic and the best result obtained by any single heuristic on the same problem. The same concept was successfully applied to timetabling problems using graph colouring heuristics in (Ross et al, 2004; Ross and Marín-Blázquez, 2005), 2D cutting stock problems in (Terashima-Marín et al, 2006) and constraint satisfaction problems in (Terashima-Marín et al, 2008). The 2D strip packing problem was tackled in a similar manner by Garrido and Riff (2007a; 2007b) but their hyper-heuristic was online so the evolved hyper-heuristic rules could only be used to solve the same problem instance that was used for training.

Thabtah and Cowling (2008) suggested mining hyper-heuristic rules from the solution run of Peckish (Cowling and Chakhlevitch, 2003) – a robust but slow hyper-heuristic. The data set to be mined would contain all the improving moves of Peckish. For each improving move, the data set would contain the heuristic applied as well as the features of the problem state before the heuristic was applied. Data mining could then yield hyper-heuristic rules that would dictate which heuristic to use according to the features of the current problem state. The authors suggest that such rules could replace the slower Peckish hyper-heuristic. Burke et al (2008c) also proposed using data mining to speed up a hyper-heuristic. This time, data mining is used to skip the step of calculating the objective function value of the candidate solution at each iteration. To achieve this, the authors proposed a system that, after training, can recognize patterns hidden in good candidate solutions, which can then be used to classify newly obtained solutions without calculating their fitness value.

2.1.5 Genetic Programming Hyper-Heuristics

Genetic programming was used to evolve priority dispatching rules for machine scheduling problems in (Dimopoulos and Zalzala, 2001; Ho and Tay, 2005; Geiger et al, 2006). Geiger et al (2006) presented a genetic programming system called SCRUPLES (Scheduling Rule Discovery and Parallel Learning System) to carry out a number of experiments with training sets containing problem instances of various sizes. It was observed that the generalizing ability of the learned rules improved as the number of training instances approached 10 but then worsened after 10. Tay and Ho (2008) extended their earlier work so as to evolve dispatching rules for multi-objective job-shop problems.

Genetic programming has also been used to construct heuristics for the minimisation of binary decision diagrams (Schmiedle et al, 2002), to aid compiler optimization (Stephenson et al, 2003), to solve the travelling salesman problem (Oltean and Dumitrescu, 2004; Keller and Poli 2007a; 2007b), parallel machine scheduling problems (Jakobovic et al, 2007) and a biobjective knapsack problem (Kumar et al, 2008). Fukunaga (2002) proposed a system called CLASS to evolve algorithms for solving satisfiability testing problems. CLASS maintains a population of such algorithms over several cycles and uses a composition operator to create new algorithms at the end of each cycle. The performance of CLASS was improved in (Fukunaga, 2004) by limiting the size of the trees representing the candidate algorithms. Heuristics for the same kind of problems were evolved by Bader-El-Din and Poli (2007) using a grammar-based genetic programming system.

Tavares et al (2004) used genetic programming to evolve evolutionary algorithms that solve a function optimization problem. The hyper-heuristic works by evolving an effective mapping function that maps a genotype to a phenotype. Oltean (2005) also presented a hyper-heuristic that produces evolutionary algorithms using genetic programming. The system works by treating an array of data as a population where each array member is called a register. Genetic programming is used to evolve instructions that operate on these registers in the same way genetic operators work on a population. The system was successfully applied to function optimisation, a travelling salesman problem and a quadratic assignment problem.

Burke et al (2006a) used tree-based genetic programming to automate the construction of heuristics for online bin packing problems. It was noted that in most of the runs, the genetic program managed to evolve some kind of variant of the well-known, human-designed First-Fit algorithm (Johnson et al, 1974). This work was followed by Poli et al (2007) who used a linear genetic programming system to construct heuristics for offline bin packing problems.

In (Burke et al, 2008d), genetic programming was used to evolve disposable heuristics that work out the score for each possible allocation in a 2D strip packing problem so that the allocation with the highest score is chosen at each step of the algorithm.

2.1.6 Hyper-Heuristics for Machine Learning

In Abe and Yamaguchi (2004) an inductive learning system called CAMLET (Suyama et al, 1998) is used for meta-learning. In this work, CAMLET constructs classification algorithms from a repository of methods according to a given data set. The repository of methods consists of various building block components from a selection of inductive learning methods. A control structure is then used to describe the relationship between these building blocks. Delibasic et al (2011) proposed a platform for storing reusable decision tree algorithm components such as attribute selection, split evaluation, stopping criteria and pruning strategies. A component-based framework called WhiBo is then used to recombine these components into complete decision tree building algorithms.

Pappa and Freitas (2006) used grammar-based genetic programming to evolve rule induction algorithms, having presented the original idea in Pappa and Freitas (2004). A broad category of rule induction algorithms operate via sequential covering: an initial rule is generated, covering some of the dataset, and additional rules are generated in order until the entire dataset is covered. There are several alternative ways to generate the initial and subsequent rules. For instance, we may either start with a very general high-coverage (but low accuracy) rule, and add conditions until accuracy and/or coverage move beyond a threshold. Or, we may start with a very precise rule and gradually remove conditions. In Pappa and Freitas (2006), the encoding covered a vast space of possible ways to organize this process. Pappa and Freitas (2007) shows how the same concept can be applied for multi-objective genetic programming.

Barros et al. (2011) recently proposed the idea of using genetic programming to evolve decision tree induction algorithms but no implementation has been reported yet.

2.1.7 Other Hyper-heuristics

Cowling et al (2002b) proposed using a genetic algorithm as a heuristic-selector for a scheduling problem. Each individual represents a sequence of heuristics and the heuristics encoded by the best individual of each generation are applied to the solution. Ahmadi et al (2003) used a variable neighbourhood search algorithm to find sequences of parameterized constructive heuristics to solve examination timetabling problems. A variant of the same algorithm was presented in Qu and Burke (2005). In this work, the authors highlighted the difference between working in the heuristic search space and solution search space. Two heuristic lists that have very little differences between them can result in two very different timetables. So the same effort within the high level searching is capable of exploring a much larger part of the solution space than a similar amount of effort used by local search based methods applied directly to solutions.

Ayob et al (2003) presented a novel heuristic acceptance criteria called the Exponential Monte Carlo with Counter method for hyper-heuristics that work by iteratively modifying a complete solution using a pool of low-level heuristics. This method works by always accepting solutions that improve on the previous one while the probability of accepting non-improving solutions decreases as time passes but increases as the number of non-improvement iterations increases. Another novel heuristic acceptance criterion called the Great Deluge was proposed by Kendall and Mohammad (2004a). This mechanism only accepts improved solutions and tends to favour diverse solutions as more iterations of the program are carried out. This technique was used in a hyper-heuristic to solve the channel assignment problem for cellular communication. Burke and Bykov (2008) proposed a memory-based heuristic acceptance mechanism that accepts a new solution by comparing its objective function value with that of L solutions ago. The motivation behind this technique is to allow for some worsening moves thus helping to avoid local minima, while still making "intelligent" use of information collected during search.

Asmuni et al (2005) used a fuzzy hyper-heuristic system to solve course timetabling problems. The system uses fuzzy weights to decide which event to schedule while constructing the timetable. A variant of exhaustive search is used to discover a good shape for the fuzzy membership functions. Bai and Kendall (2005) proposed a hyper-heuristic based on simulated annealing to solve a space allocation problem. This hyper-heuristic was later applied by Bai et al (2006) for timetabling problems. Burke et al (2005b) presented the ant colony algorithm (Dorigo et al, 1991) as a hyper-heuristic to solve a real world scheduling problem. The same algorithm was later used by Chen et al (2007) to solve a sport timetabling problem. Burke et al (2006b) presented a case-based system for heuristic selection to solve timetabling problems.

In Vázquez-Rodríguez et al (2007a), a genetic algorithm was used to solve the first stage of a scheduling problem after which a hyper-heuristic was used to continue working on the problem for the second stage. The hyper-heuristic itself consisted of a combination of various dispatching rules evolved through the use of a genetic algorithm. Bhanu and Gopalan (2008) presented a hyper-heuristic that works on top of three meta-heuristics to solve a grid resource scheduling problem. The hyper-heuristic works above the meta-heuristics and the meta-heuristics work on candidate schedules. The three meta-heuristics are a genetic algorithm combined with local search, a genetic algorithm with a mutation operator inspired by simulated annealing and a genetic algorithm that uses a tabu list.

2.2 Decision Trees

2.2.1 Decision Trees for Classification

A decision tree is a tree structure that can be used to classify data instances into some target class (Freitas, 2002). The tree structure is made up of attribute names at the nodes, attribute values at the branches and target classes at the leaves (see Figure 1.2). When a new instance needs to be classified, the values of its attributes are tested against the attributes found in the nodes of the tree, starting from the single topmost node and going down to the bottom. The branch taken is the one whose attribute value matches the one in the instance. The tree is traversed in this way until a leaf is met – the class at the leaf would be the class assigned to that instance.

Decision trees have been successfully used in virtually any field that involves some form of data mining. Using decision trees for data mining comes with many advantages: a decision tree is very easy to read and understand by humans as long as the tree is not too large, the most important attributes for classification can be easily identified as the ones towards the top of the tree, most decision tree building algorithms are non-parametric and they do not require an expert in domain of the data that is being mined.

2.2.2 Algorithms for Building Classification Trees

The majority of decision tree-building algorithms build decision trees in a recursive manner, using a greedy, divide-and-conquer method, by adding one attribute at a time to a growing tree. Such an algorithm works in a top-down fashion, starting with the full training data, partitioning it into smaller subsets by choosing a splitting attribute, and then recursing this partitioning process on each of the subsets. This procedure stops when all the instances in the partition belong to the same class or when the partition satisfies some stopping condition.

Stopping conditions are used to produce trees that are not overly complex and that do not overfit the training data. Overfitting happens when the classifier captures noisy data and/or spurious relationships (Freitas, 2002). Noisy data refers to errors in the training data whilst spurious relationships are relationships between attributes in the training data that are not statistically significant enough to help us in classifying future unseen data instances. Another effective way of dealing with this problem is by using a pruning technique. This involves growing a full-sized tree that perfectly fits the training data after which the pruning algorithm is used to reduce the size of the tree by removing certain nodes and branches that are believed to be insignificant to the predictive accuracy of the whole decision tree.

Morgan and Sondquist (1963) presented one of the earliest works on decision tree building algorithms. They proposed a system called AID (Automatic Interaction Detection) that finds binary splits on ordinal and nominal attributes that most reduce the sum of squares of an interval target from its mean. Lookahead split searches were allowed whereas cases with missing values were excluded. The algorithm stopped splitting the data when the reduction in sum of squares is less than some constant multiplied by the overall sum of squares. Other similar decision tree induction programs followed such as MAID (Gillo, 1972), THAID (Morgan and Messenger, 1973) and CHAID (Kass, 1980). CHAID (CHi-squared Automatic Interaction Detector) is a decision tree induction algorithm that finds non-binary splits based on adjusted significance testing. When building a tree, splitting attributes are chosen using the chi-square criterion. Splitting stops when the smallest adjusted p-value is greater than some user-defined threshold.

Breiman et al (1984) proposed the well-known CART. This decision tree building algorithm uses the Gini index for finding binary splits that yield the "purest" partitions with respect to the classes. Breiman et al noted that the Gini index can yield splits that are unbalanced with respect to size if the number of classes is large. In such cases, it was suggested that the twoing rule should be used instead of the Gini index. CART does not use stopping rules when building trees, instead it prunes fully grown trees using a technique called cost complexity pruning that requires the use of a pruning data set separate from the training set used for growing the decision tree. CART can also build trees with nodes made up of linear combinations of attributes. These combinations are found using a hill climbing method which slows down the building process of the decision tree. Other disadvantages of having such combinations of attributes in the decision tree nodes is that such nodes would be harder to interpret and they ultimately do not guarantee a more accurate decision tree. This method for finding multivariate splits was later extended by Murthy et al (1993) to include randomization for escaping local minima.

ID3 is another well-known algorithm that uses top-down induction for building decision trees. This was first proposed by Quinlan in 1979 but the last and definitive version was presented in (Quinlan, 1986). Quinlan argued that since there exist multiple trees that can classify all the objects in the training set as accurately as possible, the theory of Occam's Razor dictates that one should go for the simplest tree as it is more likely to capture some meaningful relationship between an object's class and the values of its attributes. For this reason, a simple tree would be expected to have a higher probability of classifying correctly objects outside the training set than a more complex one. This argument was later refuted by Domingos (1998) who presented various theoretical arguments supported by empirical evidence that though simplicity itself might be a desirable attribute, it does not guarantee greater accuracy. Nevertheless, Quinlan first suggested using the information gain criterion as an attribute selection measure as it will always tend to yield very simple trees. Various strategies for dealing with missing values in both the training set and test set were discussed. The chi-square measure was suggested as a stopping criterion when building trees. Quinlan (1987) later suggested using reduced error pruning instead of stopping for producing trees with better accuracy.

Quinlan also recognized that the information gain measure can unjustly favour attributes with many values. For this reason he recommended using the gain ratio criterion instead. ASSISTANT (Cestnik et al, 1987) overcame this problem by restricting the tree to binary splits only. Cestnik et al argued that the process of merging different attribute values for binary splitting helped the information gain criterion overcome the problem of favouring attributes with many values. Norton (1989) presented the IDX algorithm. This is like the ID3 algorithm with the added capability of performing lookahead when building trees so that it uses information gain to choose combinations of nodes on different levels of the tree as opposed to just one node at a time. Pal et al (1997) proposed using a genetic algorithm to fine-tune the nodes of a decision tree after it is built by an ID3-style algorithm. Their method, called RID3, was presented as an alternative to ID3 for problems that work on real data. Esmeir and Markovitch (2006) presented an alternative to RID3 called LSID3. The proposed learner uses a lookahead technique that tries to estimate the size of the smallest possible tree after splitting with each attribute. The attribute chosen to split the data is the one that yields the smallest tree.

Quinlan extended his ID3 algorithm to create one of the most widely-used classifiers: C4.5 (Quinlan, 1993). This program builds trees using the gain ratio criterion. It can also handle continuous attributes by choosing a threshold so as to split the attribute into two discrete sets: one containing values above the threshold and another containing values below or equal to the threshold. C4.5

can also be configured to consider attribute costs when building a tree. A post-pruning technique called error-based pruning is preferred to stopping as it is argued that this produces more reliable trees. This pruning technique does not make use of a separate pruning set, instead it prunes trees by replacing nodes with leaves if it improves the upper confidence limit of the predicted error rate. The C4.5 algorithm was later revised and extended in Quinlan (1998), the result of which was the commercial C5.0 algorithm. Amongst many other things, C5.0 improved the speed at which the decision trees are built, the memory usage as well as the size of the resultant decision trees.

Loh and Shih (1997) presented QUEST (Quick, Unbiased, Efficient, Statistical Tree), a binary-split decision tree algorithm that extends an earlier algorithm called FACT (Loh and Vanichsetakul, 1988). QUEST uses an unbiased attribute selection method in terms of number of values per attribute. For creating a node, it uses the F-statistic to first choose which attribute to split on. This is done before any of the attributes are actually split, giving QUEST an edge in terms of CPU time when compared to algorithms like CART that try to split each and every attribute before choosing one. QUEST then performs linear discriminant analysis for choosing split points, also allowing linear combinatons of attributes in a node. Kim and Loh (2001) extended this work to produce CRUISE (Classification Rule with Unbiased Interaction Selection and Estimation), a decision tree algorithm capable of producing multi-way splits.

2.2.3 Splitting Criteria

The literature contains several instances of decision-tree-building algorithms that use a criterion based on Shannon's entropy (Quinlan, 1986) for choosing splitting attributes. Some of them build trees by choosing splits that maximize gain in global mutual information of the whole tree. Such algorithms that explore this concept for pattern recognition are presented in Glesser and

Collen (1972), Sethi and Sarvarayudu (1982) and Talmon (1986). Other algorithms use Shannon's entropy to maximize information gain locally at each individual node. This has been applied to sequential fault diagnosis (Varshney et al, 1982), pattern recognition (Hartmann et al, 1982; Wang and Suen, 1984; Casey and Nagy, 1984; Hanisch, 1990) and machine learning (Quinlan, 1986). The G-statistic was introduced by Mingers (1987) as a replacement for the information gain metric in ID3 (Quinlan, 1986). This metric is based on information theory and is a very close approximation to the chi-square distribution. Baim (1988) developed a measure based on information theory called relevance and used it to create trees for a complex medical problem.

Mántaras (1991) proposed using symmetric gain ratio as an alternative to the gain ratio criterion for choosing splits. This method tries to minimize the distance between the data set partitioned using the class values and the data set partitioned using the splitting attribute values. Mántaras also gave a formal proof that this measure is efficient in compensating for information gain's bias towards attributes with a large number of values. Merckt (1993) proposed a splitting metric that combined information gain with geometric distance so as to choose splits that yield dense sets that are far apart. The J-meausure was first introduced by Smyth and Goodman (1991) to estimate the information content of a rule. This was later adapted as a measure for choosing splitting attributes by Kononenko (1995). Abellán and Moral (2003; 2005) presented the IIG criterion (Imprecise Information Gain). This measure calculates the maximum entropy of convex sets of probability distributions. Marques de Sá et al (2009) use a measure that used a minimum entropy-of-error (MEE) strategy that works with the distribution of the errors originated by the node splits.

Another class of splitting criteria are based on the distance between class probability distributions. The Kolmogorov-Smirnoff distance was used in (Friedman, 1977; Rounds, 1980) for growing trees for two-class problems after which it was applied by Haskell and Noui-Mehidi (1991) to multi-class problems. The chi-square statistic has been used as a splitting criterion in (Hart, 1984; White and Liu, 1994). A distance-based measure called Bhattacharya distance was used in (Lin and Fu, 1983). Apart from CART (Breiman et al, 1984), the Gini index has also been used to construct trees in (Pattipati and Alexandridis, 1990) for sequential fault diagnosis and in (Gelfand et al, 1991) for pattern recognition. Zhou and Dillon (1991) extended the Gini index to create a splitting criterion for multiway splits called symmetric Gini. The authors argued that this measure, unlike the original Gini index, does not favour attributes with many values. The separabililty of features for different classes was used a basis for the splitting criteria used in (Fayyad and Irani, 1992; Zhengou and Yan, 1993). Kira and Rendell (1992) presented a splitting criterion for two-class data sets called RELIEF that works by trying to measure how well values of the same attribute are able to distinguish among instances that are close to each other. After comparing RELIEF (Kira and Rendell, 1992) to the Gini index, Konenko (1994) extended RELIEF to create a modified Gini index criterion that can deal with multi-class data sets. Cieslak and Chalwa (2008) proposed a splitting criterion based on the Hellinger distance metric. This criterion was the result of the authors investigation of the DKM measured proposed in (Dietterich et al, 1996). Chandra et al (2010) proposed a splitting measure called DCSM (Distinct Class based Splitting Measure) that tries to minimize the number of distinct classes in each split so as to find the split that yields the purest partitions.

There exist other splitting criteria that do not quite fall into any of the two classes mentioned above. One example is the criteria used in Moret et al (1980) and Miyakawa (1989) which uses the activity of an attribute. This measure takes into consideration the cost of testing an attribute as well as the probability that it will be tested. Kurzynski (1983; 1988; 1989) proposed an attribute selection method that takes into account the probability distribution of the training data so as to minimize the overall error probability. Grewe and

Kak (1995) also built decision trees using the probability distribution of attributes. Li and Dubes (1986) used a permutation statistic for finding binary splits when growing trees. Gaussian distribution based criteria were used in Luo et al (1987) and Wan et al (2006). Quinlan and Rivest (1989) proposed using the minimum description length statistic (Risannen, 1989) for growing trees as well as pruning them. The principle behind this criterion is to build trees that minimize training errors while keeping the size and complexity of trees to a minimum. Mehta et al (1995) used the same principle for pruning. Michie (1989) developed a measure called weight of evidence that is based on plausibility as an alternative to entropy from information theory. Kalkanis (1993) proposed using the upper bounds of the confidence interval estimate for the misclassification error of the classifier when choosing a splitting attribute. This is in contrast to measures like information gain and Gini index that do not consider a worse goodness value after splitting. Algorithms that make use of the number of misclassified points in their attribute selection criterion were proposed in (Heath et al, 1993; Lubinsky, 1993; 1994; Murthy et al, 1994). Azam et al (2007) use an impurity measure based on the exponent function for choosing good splits.

2.3 Summary

A significant part of the literature on decision tree building algorithms is dedicated to the splitting heuristic employed by these algorithms. It seems that the splitting heuristic is crucial to the performance of the resultant decision tree. Furthermore, there is no one splitting heuristic that is universally better - an issue that will be discussed in more detail in the following chapter. What seems to be lacking is a system for choosing which heuristic to use according to the problem at hand. We believe that hyper-heuristics would provide the perfect framework for automating this process. Hyper-heuristics have been successfully applied to a wide variety of problems.

Yet, to the best of our knowledge, there has been only one attempt at applying hyper-heuristics for discovering new data mining algorithms. Pappa and Freitas (2006) used genetic programming for creating novel rule induction algorithms. Barros et al (2011) also discuss a hyper-heuristic for designing novel decision tree induction algorithms but so far no implementation of the system has been reported. Given the promise hyper-heuristics have shown in Pappa and Freitas (2006), as well as other areas, we believe that they are worth considering for decision-tree based data mining.

Chapter 3

Hyper-heuristic Rules using Partition Size

3.1 Background

3.1.1. Splitting Criteria for Building Decision Trees

Imagine a dataset held by a loan company with the attributes “gender”, “salary”, “age” and the class attribute “high-risk”, where the values for high-risk are either “yes” or “no” based on past experience. A decision tree for this data may look like the one in Figure 3.1. When we start to build a tree, the first decision to make is the choice of attribute for the root node. The key to this choice is to examine, for each attribute, how well it divides the data in terms of the target class. For example, if we found that all males were high risk and all females were low risk in the training data set, then “gender” is a perfect attribute to split on. If instead we found that male and female instances contained equal proportions of high and low risk cases, then we seem to gain nothing by splitting on gender.

Notice that, once we have chosen the attribute for the root node, we create a child node for each value of that attribute. In the example in Figure 3.1, all of the “male” data instances are carried to the left hand child, and all of the “female” instances are carried to the right-hand child. For each of these nodes, we now have the same decision to make, and will again use a heuristic to decide which attribute to split on. However, the difference is that each node “carries” a specific set of instances, and the heuristic scores will therefore depend on the position of the node in the tree. We illustrate the pseudo-code for this algorithm in Figure 3.2.

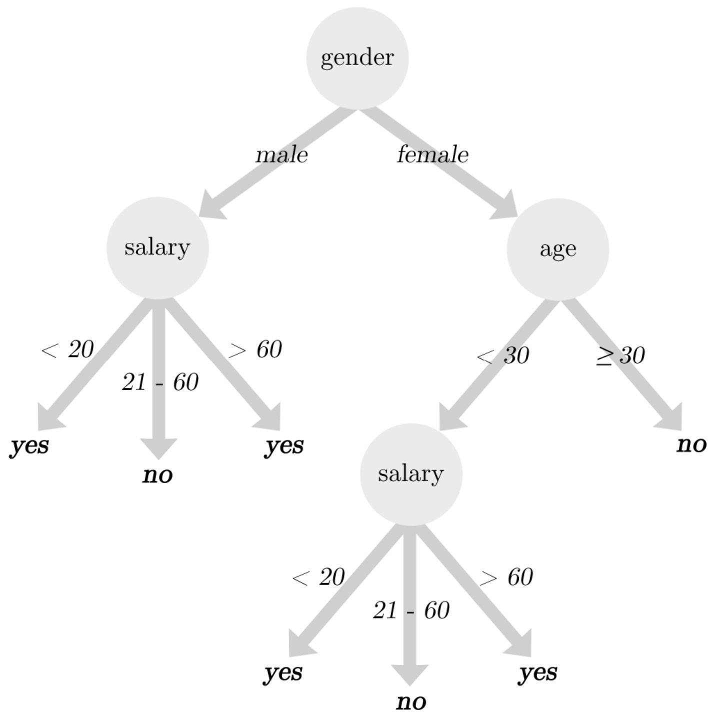

Figure 3.1: An Example of a Decision Tree

One of the most well-known decision tree building algorithms that works in this manner is ID3 (Quinlan, 1986). When building a decision tree, ID3 uses *information gain* to choose which attribute to use for creating a node in the tree. More precisely, before creating a node ID3 calculates the information gain for each candidate splitting attribute, after which the attribute with the largest information gain is chosen for the tree node. This is equivalent to making **h** in the pseudo-code in Figure 3.2 equal to information gain.

Quinlan chose to work with information gain at the time as he argued that this heuristic tends to produce small trees and, using the principle of Occam's Razor, simpler trees should be preferred over more complex ones as they are more likely to capture some meaningful relationship between an object's class and the values of its attributes (Quinlan, 1986).

```
Let D = our initial data partition /*the whole training set*/
Let A = list of all attributes in D
Let h = some heuristic that sorts a list of attributes
Let T = <empty tree>
Let n = <empty tree node>

Insert n in T
CALL GrowTree(D, A, h, n)

GrowTree(D, A, h, n)
    IF <all instances in D are of the same class> THEN
        c = class of instances in D
        Create leaf at n with label c
    ELSE
        Sort A using heuristic h
        a = attribute ranked first in A /*the splitting attribute*/
        Use attribute a for node n
        V = set of all distinct values a can have
        FOR EACH v in V
            Create a branch b from n using value v
            Create a new empty child node nv at branch b
            Dv = data partition containing only instances where a = v
            Av = A minus a
            CALL GrowTree(Dv, Av, h, nv)
        END /*FOR EACH*/
    END /*IF*/
END /*GrowTree*/
```

Figure 3.2 A Typical Decision Tree Building Algorithm

The information gain of a splitting attribute can be thought of as the expected amount of information gained for the purpose of predicting the target class of a given instance, should we know beforehand the value of the splitting attribute for the same instance. The entropy of class attribute ***c*** in data partition ***d*** can be calculated as follows:

$$H(\boldsymbol{d},\ \boldsymbol{c}) = -\sum_{i=1}^{m} \left(P(\boldsymbol{d},\ \boldsymbol{c}_i) . \log_2 P(\boldsymbol{d},\ \boldsymbol{c}_i) \right)$$

where, $P(\boldsymbol{d},\ \boldsymbol{c}_i)$ is the probability of the categorical class attribute $\boldsymbol{c}$ having value $\boldsymbol{i}$ in partition $\boldsymbol{d}$,

$\boldsymbol{m}$ is the number of unique values that class attribute $\boldsymbol{c}$ can have.

Equation 3.1

The conditional entropy of class attribute $\boldsymbol{c}$ in data partition $\boldsymbol{d}$ given the value of categorical attribute $\boldsymbol{y}$ can be calculated as follows:

$$H(\boldsymbol{d},\ \boldsymbol{c} \mid \boldsymbol{y}) = -\sum_{i=1}^{n} \left(P(\boldsymbol{d},\ \boldsymbol{y}_i) .\ H(\boldsymbol{d},\ \boldsymbol{c} \mid \boldsymbol{y}_i) \right)$$

where, $P(\boldsymbol{d},\ \boldsymbol{y}_i)$ is the probability of attribute $\boldsymbol{y}$ having value $\boldsymbol{i}$ in partition $\boldsymbol{d}$,

$\boldsymbol{n}$ is the number of unique values that attribute $\boldsymbol{y}$ can have,

$H(\boldsymbol{d},\ \boldsymbol{c} \mid \boldsymbol{y}_i)$ is the entropy of class attribute $\boldsymbol{c}$ in partition $\boldsymbol{d}$ for only those instances that have attribute $\boldsymbol{y}$ with value $\boldsymbol{i}$.

Equation 3.2

We can then calculate the information gain *IG* for splitting attribute $\boldsymbol{y}$ on data partition $\boldsymbol{d}$ as follows:

$$IG(\boldsymbol{d},\ \boldsymbol{y}) = H(\boldsymbol{d},\ \boldsymbol{c}) - H(\boldsymbol{d},\ \boldsymbol{c} \mid \boldsymbol{y})$$

Equation 3.3

Information gain is just one possible heuristic for finding good splitting attributes. Another example of such a heuristic is *gain ratio*. This heuristic is used by C4.5 (Quinlan, 1993). Quinlan chose this heuristic over information gain as he argues that the information gain heuristic has a bias that favours attributes with many values. Gain ratio mitigates this by normalizing the information gain value using the entropy of the splitting attribute. Thus, the gain ratio GR for splitting attribute $\boldsymbol{y}$ on data partition $\boldsymbol{d}$ can be calculated as follows:

$$GR(\boldsymbol{d},\ \boldsymbol{y}) = \frac{IG(\boldsymbol{d},\boldsymbol{y})}{H(\boldsymbol{d},\boldsymbol{y})}$$

Equation 3.4

The heuristic that chooses which attribute to place at each node has a big effect on the make-up of the resultant decision tree. Two similar decision tree building algorithms that use different heuristics for choosing splitting attributes can produce two very different decision trees of varying complexity and predictive accuracy. The importance of this heuristic in relation to the performance of the overall algorithm is reflected by the substantial amount of research work dedicated to the creation of such heuristics as well as to the improvement of existing ones. Nine other heuristics that have been used for the same purpose are: *chi-square* (Kass, 1980), *symmetric gain ratio* (Mántaras, 1991), *gini index* (Breiman et al, 1984), *modified gini index* (Kononenko, 1994), *symmetric gini index* (Zhou and Dillon, 1991), *J-measure* (Smyth and Goodman, 1991), *minimum description length* (Quinlan and Rivest, 1989), *relevance* (Baim, 1988), *RELIEF* (Kira and Rendell, 1992) and *weight of evidence* (Michie, 1989).

3.2 Experiments Comparing Splitting Criteria

3.2.1. Background

There are numerous studies that attempted to compare the performances of two or more of these splitting criteria. Some of these studies highlighted the different biases that come with using the various splitting heuristics (Ben-Bassat, 1978; Mingers, 1987; White and Liu, 1994; Breiman, 1996; Badulescu, 2007). Some other studies suggested that different splitting heuristics tend to produce decision trees that are not too different from each other. Baker and Jain (1976) concluded that the rankings produced by various heuristics are similar while Raileanu and Stoffel (2004) discovered that information gain and the gini index only disagree on 2% of the cases after formally comparing the two heuristics. Other authors questioned the importance of the choice of the splitting heuristic. Breiman (1984) maintained that the choice of stopping rules is much more important than the choice of splitting criteria for building good decision trees. Mingers (1989) went so far as to conclude that a random attribute selection method is just as good as any other splitting criteria on the basis of experimental evidence. This was later refuted by Buntine and Niblett (1992) who demonstrated that random attribute selection is prone to build trees that overfit the training data and also performs significantly worse when noise is introduced in the training data. This conclusion was also backed by Liu and White (1994) after they conducted a number of experiments using synthetic data sets.

However, the overall consensus in the literature seems to be that there is no one heuristic that is overall better than any other heuristic when taking its average performance over many different data sets. We decided to investigate this by testing each of the 12 different splitting heuristics mentioned in 3.1.1 on 10 different data sets.

3.2.2 Experimental Details

3.2.2.1 ID3 Algorithm

We implemented and tested 12 different decision tree builders (DTBs) modelled on Quinlan's ID3 mechanism for growing trees. They are all programmed to work as described in the pseudo-code of Figure 3.2. The 12 DTBs only differ in the splitting heuristic they employ. The 12 splitting heuristics tested are:

Chi-Square (CHI)
Information Gain (IG)
Gain Ratio (GR)
Gini Index (GINI)
J-Measure (JM)
Minimum Description Length (MDL)
Modified Gini Index (MGINI)
Relevance (RLV)
Relief (RLF)
Symmetric Gain (SGAIN)
Symmetric Gini (SGINI)
Weight of Evidence (WOE)

This effectively means that we tested 12 different DTBs: DTB-CHI, DTB-IG, DTB-GR, DTB-GINI, etc. - one for each splitting heuristic.

All the DTBs allow for multi-way splits and do not group attribute values together when splitting on an attribute. This means that when a splitting attribute is chosen for a node in the decision tree, a branch for every distinct value of that attribute is created from that node. In the case of numerical attributes, we use a discretization of the attribute values that partitions the values into a small set of intervals. There are many ways to discretize (Kerber, 1992; Fayyad and Irani, 1993; Han and Kamber, 2000). In prior experiments we have found that, from the viewpoint of decision tree quality, equal-frequency-binning (Han and Kamber, 2000) with five bins performs as well as most other methods. All of the numerical attributes in the data sets used in the experiments presented in this thesis are pre-processed into five discrete categories via equal-frequency-binning. This effectively means that the smallest 20% of values of a numerical attribute are in bin 1, the next largest 20% are in bin 2, and so on. These bins might be labelled from "very small" to "very large" (for example), if we were to build an easily understandable decision tree.

Our DTBs do not perform any stopping or pruning. We realize that we would have produced more accurate and compact decision trees had we allowed for stopping or pruning but the aim of our experiments was to compare different splitting heuristics and not to produce decision trees of optimal predictive accuracy. All problems were treated as binary-classification problems. This means that the decision trees built by our algorithms can predict whether a given instance is of the target class or not. This is in contrast to multi-class classifiers that can classify instances into one of three or more classes. Treating data sets whose class attribute consists of two possible distinct values as binary classification problems was trivial: one value is chosen to be the target class while the other value is treated as the non-target class. For data sets that contain a class attribute with more than two distinct values we pick one of the values to be the target class and the rest of the values are lumped together to form the non-target class. In such cases we used the documentation that came with the data sets to guide us in choosing an appropriate target class. When such documentation was missing, we used our best judgement in choosing a class attribute.

3.2.2.2 Data Sets

The data sets we use were downloaded from the UCI Machine Learning repository (Asuncion and Newman, 2007) and their names along with their respective sizes are detailed in Table 3.1. For this set of experiments we used the following 10 data sets: **car**, **credit**, **contrac**, **derma**, **ecoli**, **flags**, **heart**, **ionosphere**, **wine** and **yeast**.

Name	Attribute Distribution		Instances	Percentage with Target Class	Target Class [target value]
	Categoric	Numeric			
car	7	0	1728	70%	car [unacc]
contrac	8	2	1473	43%	contraceptive method used [1]
credit	8	6	690	56%	A16 [-]
derma	8	1	366	17%	eryhemato-squamous disease [2]
ecoli	8	8	336	43%	localization site [cp]
flags	8	4	194	31%	religion [1]
heart	8	6	270	44%	disease present [2]
ionosphere	8	34	351	36%	class [b]
spect	8	0	267	79%	overall diagnosis [1]
votes	8	0	435	45%	party [democrat]
wine	8	14	178	39%	c2 [2]
yeast	8	9	1484	31%	protein site [CYT]

Table 3.1 Data sets

When running a DTB on a data set we use 10-fold cross validation. This involves splitting the data set into 10 different folds (partitions) of approximately equal size. The folds are created in a random manner but we make sure that the class distribution in each fold is similar to the class distribution in the entire data set. One fold is chosen as the test set and the rest of the 9 folds are used to create a training set which the DTB uses to build a decision tree. The resultant decision tree is then tested on the test set, i.e. the fold we did not use throughout training. This process is repeated 9 more times, each time using a different fold for the test set. This way, for each DTB run on a data set, we end up with 10 different accuracy values, one for each of the 10 folds. We use the mean of these 10 values as the overall accuracy value for that particular DTB run. The accuracy value for a decision tree on a test set is simply the percentage of instances in the test set that are classified correctly by the decision tree.

Each of the 12 DTBs were run 10 times on each data set, each time using a different make-up for the folds. This effectively means that for each <DTB, data set> combination we have 10 different average accuracy values, one for each of the 10 runs.

3.2.3. Results and Discussion

Tables 3.2 and 3.3 show the results of our experiments. Each numerical value represents the overall accuracy value (in the case of Table 3.2) or ranking (in the case of Table 3.3) of a DTB using a particular splitting heuristic (denoted by the column header) on a particular data set (denoted by the row header). The last row of values in Table 3.3 indicate the average ranking of each splitting heuristic over all the data sets.

	CHI	IG	GR	GINI	JM	MDL	MGINI	RLV	RLF	SGAIN	SGINI	WOE
credit	80.435	79.71	80.87	80.725	80.435	80.725	78.696	78.116	72.319	83.043	80.29	81.304
car	93.979	94.269	85.532	94.153	84.49	93.922	93.863	94.096	80.378	85.474	93.747	80.036
ecoli	91.355	91.952	91.658	91.952	93.44	91.64	91.961	92.255	88.725	92.549	91.658	90.463
heart	73.704	74.074	75.556	73.704	78.519	72.963	72.222	74.444	67.407	76.296	74.074	73.333
contrac	62.394	62.392	64.087	62.392	63.953	61.983	61.641	62.46	60.69	63.545	62.46	62.525
wine	83.66	80.85	80.882	84.804	88.758	85.294	88.203	84.216	87.059	80.882	85.359	87.092
flags	85.105	85.658	82.921	85.105	80.947	83.105	83.553	81.974	72.711	84.579	84.632	79.316
yeast	68.266	68.397	69.409	67.726	69.407	68.536	67.386	67.93	70.013	69.745	68.263	68.399
derma	91.276	92.072	93.709	90.983	92.08	90.18	89.895	92.62	90.105	92.605	90.45	91.802
ionosphere	86.627	88.056	87.762	87.476	86.071	87.762	85.77	87.746	84.349	89.476	87.198	87.77

Table 3.2 Results Comparing Different Splitting Criteria using Average Accuracy

The first thing to note from Table 3.2 is that for any given data set, different heuristics lead to decision trees with different predictive accuracies. The choice of which splitting heuristic to use *does* have an impact on the performance of the resultant classifier. The relevance of the splitting heuristic to the classification accuracy of the decision tree varies from one data set to another. For example, the difference in predictive accuracy between the worst-performing classifier and the best-performing classifier on the **yeast** data set is 2.627% while for the **car** data set this difference is of 14.233%. For data sets like **car**, choosing the appropriate splitting heuristic is of crucial importance.

	CHI	IG	GR	GINI	JM	MDL	MGINI	RLV	RLF	SGAIN	SGINI	WOE
credit	6	9	3	4	6	4	10	11	12	1	8	2
car	4	1	8	2	10	5	6	3	11	9	7	12
ecoli	10	5	7	5	1	9	4	3	12	2	7	11
heart	7	5	3	7	1	10	11	4	12	2	5	9
contrac	7	8	1	8	2	10	11	5	12	3	5	4
wine	9	12	10	7	1	6	2	8	4	10	5	3
flags	2	1	8	2	10	7	6	9	12	5	4	11
yeast	8	7	3	11	4	5	12	10	1	2	9	6
derma	7	5	1	8	4	10	12	2	11	3	9	6
ionosphere	9	2	4	7	10	4	11	6	12	1	8	3
average	6.9	5.5	4.8	6.1	4.9	7	8.5	6.1	9.9	3.8	6.7	6.7

Table 3.3 Results Comparing Different Splitting Criteria using Average Ranking

Table 3.3 demonstrates that there is no one heuristic that performs reasonably well on all the data sets we used for testing. None of the heuristics managed to rank within the top 8 for all the data sets. Each heuristic comes with its own bias for preferring one attribute over the others. Different biases are suited for different data sets. Relief fares very poorly when considering its average performance over all the data sets, however it still managed to outperform all the other heuristics on the **yeast** data set. This means that one cannot dismiss outright using such a splitting heuristic since it can prove to give the best results on data sets like **yeast**.

3.3 The Case for Hyper-heuristics for Decision Tree Induction

Hyper-heuristics work above heuristics and meta-heuristics. Their job is to decide which heuristic to apply at any given problem situation. This is in contrast to customized, highly-specialized algorithms that have been purposely built or had their parameters optimized to solve one particular instance of a problem. Such specialized algorithms will give very good results for the problem instance that they have been built for. However they will generally perform very badly when the conditions of the problem change. This makes such algorithms very expensive to build as they cannot be effectively reused on different problems and because expert knowledge is needed to tune the many parameters of such an algorithm.

Previous work on hyper-heuristics (see 2.1) highlights the need for general purpose algorithms that can be easily used on a variety of problems. They may not necessarily deliver the optimal results but their performance is at least competitive with that of a specialized algorithm. Such hyper-heuristics work by adapting the heuristics used to the current problem or problem state that they face. Hyper-heuristics have been successfully applied to many fields, including bin-packing problems, timetabling problems, scheduling problems, cutting stock problems and constraint satisfaction problems. There is now a sizeable body of work showing the effectiveness of hyper-heuristics.

As we can see from the results of the previous experiments for decision tree building algorithms (see 3.2), each splitting heuristic comes with its own bias that helps it prefer one splitting attribute over another. Some biases suit data sets of a particular make-up while other biases suit other data sets of a different nature. Trying to identify which heuristic suits which data set is not easy. There is currently no hard and fast rule that can be used to find the optimal splitting heuristic for any given data set. Indeed, making such a decision usually involves a lot of time and effort spent on experimentation, trying out different heuristics and then choosing one on the basis of empirical evidence.

One could manage a database containing information on which heuristic is optimal for a particular data set. Such a database could be updated whenever a new data set is encountered. If we have to work on a data set that is already present in the database, we only need to consult the database to find out which heuristic works best for that problem. However, such a database would be infeasible to maintain. Finding out which heuristic works best with a newly-encountered data set means running the decision tree building algorithm numerous times (at least once for each heuristic) – this is a very time-consuming task. Furthermore, the set of all possible data sets is huge when considering that new data sets are always being created and attributes are continuously being constructed, added or deleted from older data sets. Indeed, such a database would be very difficult to maintain.

We believe that hyper-heuristics can be used to overcome this problem. What we need is a general purpose decision tree building algorithm that can give competitive results on a variety of data sets. When faced with a new data set, we could employ a hyper-heuristic to select the heuristic to be used by the decision tree building algorithm. The hyper-heuristic would adapt the heuristic used according to certain statistical features of the data set. Such a hyper-heuristic could be a set of $\boldsymbol{m}$ IF-THEN rules of the form:

```
       IF ($\boldsymbol{dsf_1} = \boldsymbol{x_{1.1}}$) AND ($\boldsymbol{dsf_2} = \boldsymbol{x_{1.2}}$) ... ($\boldsymbol{dsf_n} = \boldsymbol{x_{1.n}}$) THEN use heuristic $\boldsymbol{h_1}$
ELSE   IF ($\boldsymbol{dsf_1} = \boldsymbol{x_{2.1}}$) AND ($\boldsymbol{dsf_2} = \boldsymbol{x_{2.2}}$) ... ($\boldsymbol{dsf_n} = \boldsymbol{x_{2.n}}$) THEN use heuristic $\boldsymbol{h_2}$
...
ELSE   IF ($\boldsymbol{dsf_1} = \boldsymbol{x_{m.1}}$) AND ($\boldsymbol{dsf_2} = \boldsymbol{x_{m.2}}$) ... ($\boldsymbol{dsf_n} = \boldsymbol{x_{m.n}}$) THEN use heuristic $\boldsymbol{h_m}$
```

where, $\boldsymbol{dsf_i}$ is the name of data set feature $\boldsymbol{i}$,

$\boldsymbol{x_{j.i}}$ is the value of data set feature $\boldsymbol{i}$ in rule $\boldsymbol{j}$,

$\boldsymbol{h_j}$ is the heuristic to be used if rule $\boldsymbol{j}$ is triggered.

The above rules would represent a hyper-heuristic that decides on which splitting heuristic to use in the decision tree building algorithm by measuring $\boldsymbol{n}$ features of the data set. These $\boldsymbol{n}$ features would then be compared to each of the $\boldsymbol{m}$ rules so that the heuristic chosen is the one whose rule conditional best matches the data set features. Our task would then boil down to identifying the set of $\boldsymbol{n}$ data set features that can best characterize the problem for the purposes of choosing a suitable splitting heuristic as well as discovering the $\boldsymbol{m}$ rules that could successfully guide such a hyper-heuristic towards finding a good heuristic for any given data set.

We go one step further. The way a traditional decision tree building algorithm works is by always applying the same heuristic at each step of the tree-building process (see Figure 3.2). As the algorithm works its way down the tree, the data set partition which needs to be split gets smaller and smaller. Also, as you go deeper in the tree, the set of available candidate splitting attributes changes as fewer attributes remain. Since the problem state of our decision tree building algorithm is continuously changing as the tree is being built, we see no reason why the method for choosing a splitting attribute has to be fixed throughout the whole tree-building process. Indeed, we might get better trees if we adapt the heuristic to be used according to the data set partition that needs to be split. In such a scenario, our hyper-heuristic rules would be applied to any possible data set partition (instead of just the initial complete data set). Our decision tree building algorithm would employ a toolbox of heuristics and our hyper-heuristic would then pick and choose the best heuristic according to the features of the partition that needs to be split.

The rest of this chapter goes through our initial attempts at creating such a hyper-heuristic.

3.4 Hyper-heuristic Rules that Choose Ranking

3.4.1 Problem State Representation & Choice of Splitting Method

As the decision tree building algorithm is running, our hyper-heuristic decides which method to use for choosing a splitting attribute at each node of the developing decision tree. The hyper-heuristic uses information about the current problem state, i.e. the data partition left to split, to make this decision. In our first attempt at creating such a hyper-heuristic, we represent the problem state in the hyper-heuristic rules as the number of instances left in the data partition. Recall from 3.1.1 that the data partition left to split gets smaller and smaller as we go deeper down the tree. Different methods for choosing a splitting attribute might be suitable for creating different tree nodes depending on the depth of the node in the decision tree. We therefore represent a hyper-heuristic that decides on how to split the data when faced with a data partition $\boldsymbol{d}$ as a set of 200 rules:

```
     IF ($\boldsymbol{size}_d$ = 0.5%) THEN choose attribute ranked $\boldsymbol{r}_1$th
ELSE IF ($\boldsymbol{size}_d$ = 1.0%) THEN choose attribute ranked $\boldsymbol{r}_2$th
ELSE IF ($\boldsymbol{size}_d$ = 1.5%) THEN choose attribute ranked $\boldsymbol{r}_3$th
...
ELSE IF ($\boldsymbol{size}_d$ = 100%) THEN choose attribute ranked $\boldsymbol{r}_{200}$th
```

where, $\boldsymbol{size}_d$ is the percentage of instances left in data partition $\boldsymbol{d}$,

$\boldsymbol{r}_i$ is the ranking of the attribute to use for splitting the data after sorting the set of available attributes using some heuristic $\boldsymbol{h}$ should rule

i be triggered (where each r_i has a value between 1 and 6, chosen by the genetic algorithm).

```
Let D = our initial data partition /*the whole training set*/
Let A = list of all attributes in D
Let h = some heuristic that sorts a list of attributes
Let T = <empty tree>
Let n = <empty tree node>

Insert n in T
CALL GrowTree(D, A, h, n)

GrowTree(D, A, h, n) /*creates node at n in T*/
    IF <all instances in D are of the same class> THEN
        c = class of instances in D
        Create leaf at n with label c
    ELSE
        Sort A using heuristic h
        s = size of partition D
        r = GetRanking(s)
        a = attribute ranked rth in A /*the splitting attribute*/
        Use attribute a for node n
        V = set of all distinct values a can have
        FOR EACH v in V
            Create a branch b from n using value v
            Create a new empty child node nv at branch b
            Dv = data partition containing only instances where a = v
            Av = A minus a
            CALL GrowTree(Dv, Av, h, nv)
        END /*FOR EACH*/
    END /*IF*/
END /*GrowTree*/

GetRanking(s)
    Use hyper-heuristic rules to return ranking r based on s
END /*GetRanking*/
```

Figure 3.3 Hyper-heuristic Decision Tree Building Algorithm that Chooses Ranking

Whenever the decision tree building algorithm is faced with the task of splitting a data partition to create a tree node, the hyper-heuristic first calculates the size of the current partition by counting the number of instances in it. Then this value is used together with the size of the initial training data set so as to calculate the percentage of instances left in the current data partition.

In this set of experiments, our hyper-heuristic has only one heuristic available for ranking the available candidate splitting attributes. What varies from one hyper-heuristic rule to another (i.e. from one problem state to another) is the ranking of the attribute chosen to split the data. Recall that in traditional decision tree building algorithms a splitting attribute is chosen by first sorting the available attributes into a list using some heuristic, after which the first-ranked attribute is chosen as the splitting attribute. For our first set of hyper-heuristic experiments we decided to keep the sorting heuristic fixed while only varying the ranking of the chosen splitting attribute from the sorted list of attributes. Note that heuristic $\boldsymbol{h}$ never changes. Figure 3.3 contains the pseudo-code for our first hyper-heuristic decision tree building algorithm.
We tried two sets of experiments using this type of hyper-heuristic, one set uses information gain to sort the attributes while the other set uses gain ratio.

3.4.2 Searching for Good Hyper-heuristics using Genetic Algorithms

We use a genetic algorithm to search for good values for the rankings ($\boldsymbol{r_1}$, $\boldsymbol{r_2}$, $\boldsymbol{r_3}$, ..., $\boldsymbol{r_{200}}$) in our hyper-heuristic rules. Genetic algorithms are popular search optimization methods inspired by Charles Darwin's theory of evolution (Holland, 1975). A typical genetic algorithm maintains a population of candidate solutions at any one time. The initial population is created in a random manner after which other populations are evolved iteratively over a series of generations using three operators: selection, crossover and mutation. The selection operator works on the entire population and its job is to identify which individuals in the population will pass on their "genetic material" to the next generation's population. Central to the effectiveness of the selection operator is the fitness function that evaluates how good a solution each individual represents – the better the solution, the higher the probability of that individual's genes being passed on to the next generation.

The crossover operator works on two individuals in the population by swapping some of their "genetic material". By "genetic material" we mean atomic parts of the solution encoded by the individual. The crossover operator produces two new individuals (offspring) that inherit genes from the original two parents. The purpose of this operator is to *intensify* the search so that the genes from the best individuals in the population are combined together in the hope of creating better offspring. Intensification is the process of combining good building blocks (i.e. sub-sections of an individual's genes that contribute to a high fitness value) from different individuals. The mutation operator helps *diversify* the search by making random jumps in the search space. It achieves this by making very small random changes in the genes of an individual in the hope that it will land in a promising area of the search space previously unexplored. In this way, a genetic algorithm creates one generation after another using these three operators. This process stops after the fitness value of an individual in the population meets some requirement or after a preset number of generations. The individual with the best fitness value over all the generations is the one returned by the genetic algorithm.

One can visualize the search space of a gentic algorithm using a *fitness landscape* (Jones, 1995). A fitness landscape consists of points that represent candidate solutions. Individuals that are very similar in genetic make-up are bound to be closer to each other on the landscape. Furthermore, the higher the fitness of an individual, the bigger the height of the respective point on the landscape. Please note that we did not carry out any investigations into the fitness landscapes of our genetic algorithms.

We are searching for a good hyper-heuristic. Thus, each individual in our genetic algorithm represents a complete set of 200 hyper-heuristic rules as presented in 3.4.1. We encode these rules as an integer string of length 200: r_1, r_2, r_3, ..., r_{200}, where the first integer represents the ranking value for the first hyper-heuristic rule, the second integer represents the ranking value of the second hyper-heuristic rule, etc. Each ranking value gene was allowed to vary between 1 and 6, which means that the resultant hyper-heuristic can decide on choosing any attribute that ranks 1st to 6th. The value of 6 was chosen as the optimal lowest-ranking value on the basis of empirical evidence produced by experimentation. This involved running a series of preliminary experiments testing different ranges of allowed ranking genes. In our experiments, the range of 1-6 gave the best results.

The choice of fitness function is crucial to the performance of any genetic algorithm. The fitness function we used involves running the decision tree building algorithm using the hyper-heuristic rules encoded by the individual on one or more data sets using cross validation (as described in 3.2.2.2). The fitness value of an individual is the average predictive accuracy of the decision trees produced by the hyper-heuristic encoded in that individual.

The population was of size 40 and the genetic algorithm terminated after 60 generations. The number of parents chosen by the selection operator is 75% of the population size. This means that 75% of any given population is made up of offspring created from individuals selected from the previous generation while the remaining 25% are direct copies of the best individuals from the previous generation. This is a very selective genetic algorithm, deliberately so to promote fast progress, since individual fitness evaluations are quite time-consuming.

Tournament selection without replacement was used as this has been shown to be an effective method when used in other similar problems. This selection operator involves running a series of tournaments, one for each parent to be selected. A tournament is run by randomly selecting a number of individuals from the population and the individual with the highest fitness value from this set is chosen as the winner. The number of individuals competing in a tournament, i.e. the tournament size, was set to be 40% of the population size. Once an individual has been selected he is no longer eligible to participate in future tournaments – this ensures that an individual can only be selected once in one generation.

After the parents have been selected, they are paired up and one-point crossover is applied to each pair. This crossover technique involves identifying a random single crossover point so that all the genes to one side of this point are swapped between parents. We also did some preliminary tests comparing this crossover technique to two-point crossover and uniform crossover. Our tests showed that one-point crossover has a slight edge over the other two crossover methods. Point mutation is then applied to each offspring produced by crossover. When applied to an individual, this mutation operator goes through each gene of the individual and randomly changes the value of the gene with a very small probability. In our case, mutation was applied with a probability of 5%. This figure was chosen on the basis of empirical evidence produced by experimentation.

We also carried out experiments comparing the search results produced by our genetic algorithm to a simple hill-climbing method and to a genetic algorithm that does not employ a crossover method. Results suggested that our genetic algorithm is somewhat better at discovering hyper-heuristic rule sets than the other two search methods.

3.4.3 Data Sets

For these experiments we use data sets **car**, **contrac**, **credit** and **votes** (see Table 3.1). We run two sets of experiments: the first set uses *single* data sets while the second set uses *multiple* data sets. The single data set experiments use only one data set which is split into two halves: one half is used for training the hyper-heuristic – called the *hyper-heuristic training set* (HHTRS) – and the other half is used for testing the hyper-heuristic produced by the training phase – called the *hyper-heuristic test set* (HHTES). The training phase we refer to here is actually the search conducted by the genetic algorithm. The HHTRS is the data set used by the fitness function of the genetic algorithm, hence it is the data set used to "learn" the hyper-heuristic. The fitness value of a candidate hyper-heuristic is the average predictive accuracy of the decision trees produced by that hyper-heuristic after running it on the HHTRS using 10-fold cross validation. This means that the HHTRS is split into a *building set* (for building a decision tree) and a *validation set* (for testing the resultant decision tree) 10 times, each time using a different validation set and building set.

The HHTES is not used by the genetic algorithm in any way and so we are free to use it for testing our resultant hyper-heuristic. We test the resultant hyper-heuristic by using it to create a decision tree on the HHTRS which is then tested on the HHTES using 10-fold cross validation. The performance of the hyper-heuristic is measured by the average predictive accuracy it achieves on the HHTES. We compare our hyper-heuristics to traditional decision-tree building algorithms by running normal ID3 on the same HHTES using the same cross-validation method. We ran four such experiments: one for each of the data sets.

The experiments that use multiple data sets work in the same manner with the difference that the hyper-heuristic is trained and tested on more than one data set. This means that our HHTRS would comprise of several halves of data sets while the HHTES would be made up of the other halves of the same data sets. In this case, the fitness value would be the average performance of the hyper-heuristic over all the data sets in the HHTRS. Likewise, the performance measure of our resultant hyper-heuristic is the average result it obtains over all the data sets in the HHTES. We ran one such experiment using all four data sets.

The motives for running these two sets of experiments are different from each other. In the case of the experiments using single data sets, we expect the resultant hyper-heuristic to be highly-optimized for the data set used during training. One would hope that it manages to outperform decision tree building algorithms that use static, non-adaptive heuristics on this same data set. We call this a *specialized* hyper-heuristic. In the case of the experiments using multiple data sets, we expect the resultant hyper-heuristic to be a good overall algorithm that can be reliably applied to any one of the data sets used for training. We do not expect such a hyper-heuristic to outperform traditional single-heuristic algorithms on each and every data set used for training the hyper-heuristic. However we do expect it to perform consistently well on each of the data sets it was trained on. We call this a *generalized* hyper-heuristic.

We run each set of experiments 10 times, each time varying the way we split the data set into the HHTRS and the HHTES as well as the seeds of the initial population of the genetic algorithm. We present the average result for each experiment.

3.4.4 Results

3.4.4.1 Single Data Set Experiments

Table 3.4 shows the results for the single data set experiments using the information gain heuristic while Table 3.5 shows the results for the same kind of experiments using the gain ratio heuristic. The values under column HH represent predictive accuracy values achieved by our hyper-heuristic while those under column DTBA represent predictive accuracy values achieved by a regular decision tree building algorithm. This decision tree building algorithm works in the manner described in 3.2.2.1 (i.e. it always chooses the first ranked attribute when splitting the data) and it uses the same heuristic as the one used by the hyper-heuristic. This means that the DTBA in Table 3.4 uses information gain while the one in Table 3.5 uses gain ratio. A predictive accuracy value is the average percentage of correctly classified instances. The third column represents the p-values worked out using a Student's two-tailed t-test.

Using Information Gain			
	HH	DTBA	*p-value*
car	93.296	93.852	*0.291*
contrac	60.942	62.12	*0.109*
credit	77.492	78.35	*0.284*
votes	92.994	93.416	*0.592*

Table 3.4 Predictive Accuracy of Single Data Set Experiments Using Information Gain

Using Gain Ratio			
	HH	DTBA	*p-value*
car	92.999	94.122	*0.116*
contrac	62.187	62.672	*0.507*
credit	79.213	79.594	*0.796*
votes	92.857	92.859	*0.998*

Table 3.5 Predictive Accuracy of Single Data Set Experiments Using Gain Ratio

None of the hyper-heuristics managed to significantly outperform the traditional DTBAs on any of the data sets. All of the hyper-heuristics managed to get very competitive results, notably the hyper-heuristic running on the **votes** data set using the gain ratio heuristic. However, assuming a standard confidence level of 90%, none of the hyper-heuristic methods were signficantly different from the standard methods on any of the data sets.

3.4.4.2 Multiple Data Set Experiments

Tables 3.5 and 3.6 display results for the multiple data set experiments using information gain and gain ratio respectively.

Using Information Gain			
	HH	DTBA	*p-value*
car	92.847	93.797	*0.046*
contrac	61.802	61.706	*0.883*
credit	78.961	78.724	*0.887*
votes	93.089	92.961	*0.915*

Table 3.6 Predictive Accuracy of Multiple Data Set Experiments Using Information Gain

Using Gain Ratio			
	HH	DTBA	*p-value*
car	92.733	93.275	*0.329*
contrac	62.836	63.855	*0.260*
credit	79.478	79.566	*0.958*
votes	92.662	92.193	*0.503*

Table 3.7 Predictive Accuracy of Multiple Data Set Experiments Using Gain Ratio

The generalized hyper-heuristics methods never managed to perform significantly better than the standard DTBAs. This means that there is no benefit in using a hyper-heuristic for this group of data sets. However, it is interesting to note that when comparing the results of the hyper-heuristic to the DTBA on each data set individually, the hyper-heuristic seemed to achieve slightly better results in four instances. This is true for the hyper-heuristic that uses information gain in the case of data sets **contrac**, **credit** and **votes** as well as for the hyper-heuristic that uses the gain ratio heuristic in the case of the **votes** data set. However it turns out that none of these differences are statistically significant.

3.4.5 Discussion

With the single data set experiments we tried to evolve specialized hyper-heuristics trained for data sets from a particular domain. The results we got suggest that we failed to do this. None of the specialized hyper-heuristics managed to outperform the regular decision tree building algorithms. It seems that our choice of problem state representation for the hyper-heuristic rules is not so effective. These hyper-heuristics failed to find a link between the number of instances within a data partition and the best way to split that same partition. The same holds for the multiple data set experiments. Our aim in this case was to evolve generalized hyper-heuristics that need not always outperform standard algorithms but at least give a better overall performance over the set of data sets they were trained on. Again, the way we represent the problem state in our hyper-heuristic rules may have lead to these experiments giving disappointing results.

It is interesting to note that even though the hyper-heuristics that were trained on multiple data sets did not outperform the regular algorithms when considering their overall average result, they did actually manage to achieve a slight improvement on half of the individual data sets. This suggests that there might be such a thing as hyper-heuristics for decision tree building algorithms that can improve over regular algorithms by adapting the splitting heuristic used. One of the reasons why the specialized hyper-heuristics may have failed to achieve this could be because they overfit the particular make-up of the training set they were evolved on. The "extra" data sets used in the training phase of the multiple data sets experiments could have helped prevent the hyper-heuristic from overfitting any one of the data sets.

In this chapter, we present one more hyper-heuristic that uses the same problem state representation but is given the option of choosing how to sort the candidate splitting attributes using either information gain or gain ratio.

3.5 Hyper-heuristic Rules that Choose Heuristic and Ranking

3.5.1 Problem State Representation & Choice of Splitting Method

We decided to modify our first hyper-heuristic and re-run the same experiments as in 3.4 using this second hyper-heuristic. This modified hyper-heuristic has two sorting heuristics available to use.

```
Let D = our initial data partition /*the whole training set*/
Let A = list of all attributes in D
Let T = <empty tree>
Let n = <empty tree node>

Insert n in T
CALL GrowTree(D, A, n)

GrowTree(D, A, n)
    IF <all instances in D are of the same class> THEN
        c = class of instances in D
        Create leaf at n with label c
    ELSE
        s = size of partition D
        <h, r> = GetHeuristicAndRanking(s)
        Sort A using heuristic h
        a = attribute ranked rth in A /*the splitting attribute*/
        Use attribute a for node n
        V = set of all distinct values a can have
        FOR EACH v in V
            Create a branch b from n using value v
            Create a new empty child node nv at branch b
            Dv = data partition containing only instances where a = v
            Av = A minus a
            CALL GrowTree (Dv, Av, nv)
        END /*FOR EACH*/
    END /*IF*/
END /*GrowTree*/

GetHeuristicAndRanking(s)
  Use hyper-heuristic rules to return ranking r and heuristic h based on s
END /*GetHeuristicAndRanking*/
```
Figure 3.4 Hyper-heuristic Decision Tree Building Algorithm that Chooses Heuristic and Ranking

The problem state representation is similar to the one described in 3.4.1 except we use 100 rules instead of 200 in the following manner:

IF ($\boldsymbol{size_d}$ = 1%) THEN use heuristic $\boldsymbol{h_1}$ & choose attribute ranked $\boldsymbol{r_1}$th
ELSE IF ($\boldsymbol{size_d}$ = 2%) THEN use heuristic $\boldsymbol{h_2}$ & choose attribute ranked $\boldsymbol{r_2}$th
ELSE IF ($\boldsymbol{size_d}$ = 3%) THEN use heuristic $\boldsymbol{h_3}$ & choose attribute ranked $\boldsymbol{r_3}$th
…

ELSE IF ($\boldsymbol{size_d}$ = 100%) THEN use heuristic $\boldsymbol{h_{100}}$ & choose attribute ranked $\boldsymbol{r_{100}}$th

where, $\boldsymbol{size_d}$ is the percentage of instances left in data partition $\boldsymbol{d}$,

$\boldsymbol{h_i}$ is the heuristic to be used for sorting the available attributes should rule $\boldsymbol{i}$ be triggered,

$\boldsymbol{r_i}$ is the ranking of the attribute to use for splitting the data after the attributes have been sorted using $\boldsymbol{h_i}$ should rule $\boldsymbol{i}$ be triggered.

When faced with a data partition of a given size, the hyper-heuristic relays back to the decision tree building algorithm two pieces of information: a) the heuristic to be used for sorting the available candidate splitting attributes and b) which attribute to choose for splitting this partition from the sorted list of candidate splitting attributes. Thus, we update the pseudo-code of our hyper-heuristic as shown in Figure 3.4.

3.5.2 Genetic Algorithm

We used the same genetic algorithm described in 3.4.2 with a different encoding for the individuals. A hyper-heuristic is now made up of 100 rules and each rule has associated with it a sorting heuristic as well as a ranking value. We encode such a hyper-heuristic as an integer string of length 200: $\boldsymbol{h_1}$, $\boldsymbol{r_1}$, $\boldsymbol{h_2}$, $\boldsymbol{r_2}$, ..., $\boldsymbol{h_{100}}$,, $\boldsymbol{r_{100}}$. Each $\boldsymbol{h_i}$ represents the sorting heuristic to be used if rule $\boldsymbol{i}$ is triggered. This gene can be either 0 or 1 where 0 represents the information gain heuristic and 1 represents the gain ratio heuristic. Each $\boldsymbol{r_i}$ value represents the ranking of the attribute to be chosen for splitting the data should rule $\boldsymbol{i}$ be triggered. The ranking gene can vary between 1 and 6 as in the previous hyper-heuristic.

As for the previous experiments, we run each set of experiments 10 times, each time varying the way we split the data set into the hyper-heuristic training set and the hyper-heuristic test set as well as the seeds of the initial population of the genetic algorithm. We report the average result for each of the 10 runs.

3.5.3 Results

Tables 3.8 and 3.9 show results for our second hyper-heuristic. The values under DTBA IG refer to the results obtained by the regular decision tree building algorithm using information gain while the values under DTBA GR refer to results obtained by the same kind of algorithm using gain ratio.

	HH	DTBA IG	*p-value*	DTBA GR	*p-value*
car	93.565	93.518	*0.904*	93.599	*0.932*
contrac	61.857	62.228	*0.540*	62.789	*0.244*
credit	78.98	79.205	*0.747*	80.212	*0.146*
votes	92.903	93.182	*0.784*	92.208	*0.516*

Table 3.8 Predictive Accuracy of Single Data Set Experiments using both Information Gain and Gain Ratio

	HH	DTBA IG	*p-value*	DTBA GR	*p-value*
car	92.674	93.566	*0.207*	93.658	*0.157*
contrac	61.178	61.014	*0.888*	61.095	*0.939*
credit	79.42	78.726	*0.527*	79.743	*0.778*
votes	91.894	93.095	*0.275*	92.584	*0.469*

Table 3.9 Predictive Accuracy of Multiple Data Sets Experiments using both Information Gain and Gain Ratio

The results of this hyper-heuristic are very similar to the results obtained by our initial hyper-heuristic. None of the hyper-heuristics managed to outperform the standard DTBAs. In fact, in all cases the hyper-heuristics' performance was not significantly different from that of the standard methods. However, the hyper-heuristic that was trained on multiple data sets did manage to get the best results for data set **contrac**. However, this difference is not statisically significant.

3.5.4 Discussion

The results achieved by the second hyper-heuristic are very similar to the ones obtained by the earlier hyper-heuristic that uses only one sorting heuristic. The specialized hyper-heuristics still seem to be overfitting the training set as not one of them managed to outperform the regular decision tree building algorithms. The overall results obtained by the "general" hyper-heuristic failed to outperform any of the regular decision tree building algorithms. However, as in the earlier experiments, they did manage to get slightly better results on one of the data sets. Though this difference is not statistically significant, we still feel that this seems to reinforce the notion that training our hyper-heuristic on multiple data sets prevents it from overfitting any one of the data sets.

3.6 Summary

In this chapter we presented our first attempts at developing hyper-heuristics for decision tree induction. These hyper-heuristics base the decision on how to split a data partition on its size. We have presented two versions of this hyper-heuristic. The first version uses a fixed heuristic for sorting candidate splitting attributes while adapting the ranking of the chosen splitting attribute. The second version is given a choice of two sorting heuristics so that it also changes the way candidate splitting attributes are sorted according to the problem state. For both versions, we tried evolving specialized hyper-heuristics that are trained on just one data set as well as generalized hyper-heuristics that are trained on a group of data sets from different domains.

All the results of our experiments suggest that there is no benefit in using such hyper-heuristics. The hyper-heuristic did not manage to find a strong correlation between the size of the partition that needs to be split and the best way of choosing a splitting attribute from that same partition. One reason for this could be that this correlation is too weak. It might well be that the size of the partition bears no relevance to the best way it should be split when building a decision tree. Another possible cause for the poor results we obtained could be the length of the chromosone representing the hyper-heuristic in the genetic algorithm. Having such a long chromosone makes it very hard for the genetic algorithm to converge to a good solution since the search space is so big. A possible improvement could be to ensure that the size of the partition varies monotonically with the value of the ranking gene when creating and manipulating the individuals of the genetic algorithm. We also noticed that during the lifetime of the genetic algorithm a considerable number of the hyper-heuristic rules were never triggered by the data partitions encountered by the decision tree building algorithms. This means that all the effort that went into evolving these genes by the genetic algorithm was wasted since these genes were never relevant to the fitness value of the individual. This issue together with the long length of the chromosone makes the search for a good hyper-heuristic a difficult task for the genetic algorithm.

Chapter 4

Hyper-heuristic Rules using Attribute Information

In chapter 3, we described hyper-heuristic rules that work within the framework of a typical decision-tree building algorithm. While building a decision tree, these rules help us decide on how to split the data at each node in the developing tree. They take into consideration the state of the current problem, i.e. the data partition to be split, so as to decide on which method to use for choosing the splitting attribute.

When designing such rules, the choice of the problem state representation as well as the available heuristics for choosing a splitting attribute are vital to the performance of the resultant hyper-heuristic. The hyper-heuristics presented in chapter 3 all characterize the problem state by looking at the number of instances left in the partition to be split. Moreover, these hyper-heuristics restricted themselves to using just two splitting heuristics: information gain and gain ratio.

In this chapter we present four alternative hyper-heuristics for the same purpose. All of these hyper-heuristics use rules that represent the problem state by some statistical property of the candidate splitting attributes that remain in the partition to be split. A bigger set of splitting heuristics is made available to these hyper-heuristics. We present experimental results for each of these hyper-heuristics. Instead of discussing the experimental results individually we present an analysis of all the results together with a discussion at the end of the chapter.

4.1 Hyper-heuristic Rules using Number of Attributes Left

4.1.1. Problem State Representation & Choice of Splitting Method

When building a decision tree, the number of attributes available to split the partition gets smaller and smaller as we go deeper down the tree. The first hyper-heuristic we present in this chapter bases its decision on how to split the data on the number of candidate splitting attributes left in the data partition to be split. Recall from 3.1 that a splitting heuristic works on a set of candidate splitting attributes by sorting them in a list after which the top-most attribute is chosen to split the data. The number of attributes left available to split the partition is an indicator of how deep we are in the tree. The effectiveness of the splitting heuristic in choosing a good splitting attribute might depend on the number of attributes it has to sort.

We therefore represent a hyper-heuristic that decides on how to split the data when faced with a data partition $\boldsymbol{d}$ as a set of $\boldsymbol{m}$ rules:

```
     IF ($attr\_left_d < x_1$) THEN use splitting heuristic $h_1$
ELSE IF ($attr\_left_d < x_2$) THEN use splitting heuristic $h_2$
...
ELSE IF ($attr\_left_d < x_{m-1}$) THEN use splitting heuristic $h_{m-1}$
ELSE use splitting heuristic $h_m$
```

where, $\boldsymbol{attr\_left_d}$ is the number of candidate splitting attributes in data partition $\boldsymbol{d}$,

$\boldsymbol{h_i}$ is the heuristic to use to sort the list of candidate splitting attributes in partition $\boldsymbol{d}$ should rule $\boldsymbol{i}$ be triggered,

$\boldsymbol{x_1, x_2, \ldots x_{m-1}}$ are integer values where $\boldsymbol{x_1} < \boldsymbol{x_2} < \boldsymbol{x_3} \ldots < \boldsymbol{x_{m-1}}$.

Using this set of hyper-heuristic rules, whenever a data partition needs to be split, we look at the number of candidate splitting attributes left in the partition to select an appropriate rule from the list of $\boldsymbol{m}$ hyper-heuristic rules. We do this by testing the conditional of each rule, one by one, starting from the top. The heuristic chosen is the one dictated by the rule whose conditional evaluates to true to the current problem state. If none of the first $\boldsymbol{m} - 1$ rules evaluate to true, we use the heuristic dictated by the default rule: $\boldsymbol{h_m}$.

Note that in this hyper-heuristic we always choose the topmost attribute for splitting the data after the set of attributes are sorted using the chosen heuristic. This holds true for all the hyper-heuristics we present from this point onwards. We present the pseudo-code for the decision-tree building algorithm that uses this hyper-heuristic in Figure 4.1.

We ran two sets of experiments using this type of hyper-heuristic. The hyper-heuristic in the first set of experiments has all 12 heuristics mentioned in 3.1.1 available to work with. These are *chi-square*, *information gain*, *gain ratio*, *gini index*, *J-measure*, *minimum description length*, *modified gini index*, *relevance*, *relief, symmetric gain*, *symmetric gini* and *weight of evidence.* The second hyper-heuristic has only 5 heuristics to work with: *information gain*, *gain ratio*, *J-measure*, *relief* and *symmetric gain.* We narrowed down the original set of 12 heuristics to this smaller set by choosing the 5 heuristics that gave the best performances on some of the individual data sets while also making sure that these heuristics produce sorted lists of attributes that are as different from each other as possible. The reason for this is that a hyper-heuristic needs a pool of effective and diverse low-level heuristics in order for it to be robust (Cowling and Chakhlevitch, 2003).

```
Let D = our initial data partition /*the whole training set*/
Let A = list of all attributes in D
Let T = <empty tree>
Let n = <empty tree node>

Insert n in T
CALL GrowTree(D, A, n)

GrowTree(D, A, n)
    IF <all instances in D are of the same class> THEN
        c = class of instances in D
        Create leaf at n with label c
    ELSE
        h = GetHeuristic(|A|) /*|A| is the number of attributes left*/
        Sort A using heuristic h
        a = attribute ranked 1st in A /*the splitting attribute*/
        Use attribute a for node n
        V = set of all distinct values a can have
        FOR EACH v in V
            Create a branch b from n using value v
            Create a new empty child node nv at branch b
            Dv = data partition containing only instances where a = v
            Av = A minus a
            CALL GrowTree(Dv, Av, nv)
        END /*FOR EACH*/
    END /*IF*/
END /*GrowTree*/

GetHeuristic(x)
    Use hyper-heuristic rules to return heuristic h based on x
END /*GetHeuristic*/
```

Figure 4.1 Hyper-heuristic Decision Tree Building Algorithm that Chooses Heuristic using Number of Attributes Left

4.1.2 Genetic Algorithm

We use a genetic algorithm to evolve our hyper-heuristic rules. An individual in the genetic algorithm represents the set of $\boldsymbol{m}$ rules described in 4.1.1. We encode these $\boldsymbol{m}$ rules as follows:

$$\boldsymbol{x_1}, \boldsymbol{h_1}, \boldsymbol{x_2}, \boldsymbol{h_2}, \boldsymbol{x_3}, \boldsymbol{h_3}, \ldots, \boldsymbol{x_{m-1}}, \boldsymbol{h_{m-1}}, \boldsymbol{h_m}$$

where for any rule i, $\boldsymbol{x_i}$ represents the $\boldsymbol{x}$ value and $\boldsymbol{h_i}$ represents the heuristic (see rule set description in 4.1.1). The $\boldsymbol{x}$ value genes are integer values that represent thresholds for the number of attributes left in the partition. Special care was taken so that whenever an individual's genes are created (when the first population is created) or manipulated (via crossover or mutation), these $\boldsymbol{x}$ values are given sensible values. This condition was met by a) making sure that none of the $\boldsymbol{x}$ value genes are smaller than 3 or bigger than the total number of attributes in the data set with the largest number of attributes, and b) respecting the condition $\boldsymbol{x_1} < \boldsymbol{x_2} < \boldsymbol{x_3} \ldots < \boldsymbol{x_{m-1}}$.

For the heuristic $\boldsymbol{h_i}$ genes we used integer values where each distinct value maps to a specific heuristic. These genes were allowed to range from 1 to 5 for the hyper-heuristics that use 5 heuristics and 1 to 12 for the hyper-heuristics that use 12 heuristics. The value of $\boldsymbol{m}$ was set to 3 so that we effectively evolved hyper-heuristic rule sets of size 3. Keeping the rule set to such a small size guaranteed chromosomes of a short length thus making it easier for the genetic algorithm to converge to a good solution.

The population was of size 45 and the genetic algorithm terminated after 100 generations. The rest of the genetic algorithm parameters were kept the same as for the one used in chapter 3. The number of offspring created after each generation was 75% of the population. Tournament selection without replacement was used with the tournament size set to 40% of the population size. One-point crossover and point mutation with a probability of 5% was employed.

4.1.3 Data Sets

We utilized the data sets mentioned in Table 3.1. We ran single data set experiments and multiple data set experiments in the same way as described in 3.4.3. The only difference is that 9-fold cross validation was used in the hyper-heuristic training phase instead of 10-fold cross validation. Reducing the number of folds makes the folds themselves bigger thus adding confidence to the results of each test fold. We ran 12 different single data set experiments, one for each of the data sets listed in Table 3.1. We also ran 3 different multiple data set experiments using 3 different combinations of data sets:

a) car, derma, ecoli, wine
b) contrac, credit, ionosphere, spect
c) yeast, votes, heart, flags

We chose these particular combinations of data sets as we wanted each set to be made up of data sets that are as different from each other as possible in terms of size, number of attributes and attribute type distribution. Each experiment was run 10-20 times, each time varying the way the hyper-heuristic training set and test set is generated as well as the random seed of the initial population of the genetic algorithm.

4.1.4. Results

4.1.4.1 Single Data Set Experiments

	car		contrac		credit		derma		ecoli		flags	
HH-5	2.938	*p-value*	5.350	*p-value*	5.250	*p-value*	3.800	*p-value*	5.550	*p-value*	5.350	*p-value*
CHI	2.188	*0.386*	8.700	*0.012*	5.950	*0.553*	4.900	*0.283*	4.300	*0.340*	5.150	*0.856*
IG	**1.375**	*0.054*	7.200	*0.100*	6.450	*0.237*	3.250	*0.598*	4.350	*0.261*	5.750	*0.734*
GR	1.875	*0.194*	**4.200**	*0.282*	5.500	*0.796*	6.650	*0.028*	5.700	*0.905*	4.000	*0.185*
GINI	1.688	*0.146*	8.000	*0.015*	6.150	*0.430*	4.550	*0.466*	4.350	*0.261*	5.750	*0.701*
JM	10.688	*0.000*	6.000	*0.585*	5.500	*0.810*	6.750	*0.013*	5.400	*0.906*	4.900	*0.669*
MDL	1.875	*0.194*	5.500	*0.899*	7.350	*0.046*	6.250	*0.016*	5.400	*0.907*	5.350	*1.000*
MGINI	2.563	*0.679*	9.050	*0.003*	6.800	*0.145*	8.050	*0.000*	4.500	*0.352*	5.350	*1.000*
RLV	3.125	*0.864*	6.700	*0.215*	6.550	*0.258*	3.000	*0.377*	4.250	*0.230*	4.500	*0.421*
RLF	12.813	*0.000*	6.200	*0.508*	12.150	*0.000*	9.850	*0.000*	9.550	*0.005*	10.250	*0.000*
SGAIN	10.250	*0.000*	4.550	*0.498*	5.250	*1.000*	**2.250**	*0.070*	**3.350**	*0.044*	5.100	*0.826*
SGINI	2.188	*0.386*	5.750	*0.698*	**4.600**	*0.511*	4.500	*0.462*	3.650	*0.072*	**3.300**	*0.037*
WOE	12.063	*0.000*	7.300	*0.135*	4.800	*0.661*	9.200	*0.000*	6.550	*0.488*	6.450	*0.385*

	heart		ionosphere		spect		votes		wine		yeast	
HH-5	4.000	*p-value*	6.250	*p-value*	3.813	*p-value*	4.600	*p-value*	5.400	*p-value*	5.300	*p-value*
CHI	6.550	*0.032*	6.400	*0.903*	5.625	*0.135*	**3.350**	*0.238*	4.000	*0.180*	6.000	*0.543*
IG	7.100	*0.006*	5.300	*0.403*	6.250	*0.087*	3.850	*0.391*	4.250	*0.278*	**4.450**	*0.431*
GR	6.200	*0.085*	5.100	*0.324*	6.813	*0.048*	**3.350**	*0.177*	**2.250**	*0.000*	6.300	*0.416*
GINI	8.100	*0.001*	5.750	*0.656*	5.375	*0.188*	3.700	*0.295*	4.600	*0.439*	5.100	*0.846*
JM	6.450	*0.043*	7.650	*0.201*	5.875	*0.119*	4.200	*0.701*	4.800	*0.584*	7.300	*0.141*
MDL	5.800	*0.129*	**4.200**	*0.078*	5.563	*0.198*	4.450	*0.874*	4.100	*0.230*	5.750	*0.718*
MGINI	8.100	*0.000*	6.050	*0.870*	8.250	*0.002*	3.850	*0.432*	6.300	*0.419*	6.000	*0.541*
RLV	6.700	*0.026*	4.900	*0.255*	4.813	*0.449*	6.450	*0.072*	4.900	*0.669*	6.250	*0.444*
RLF	3.700	*0.790*	10.100	*0.001*	4.063	*0.829*	11.250	*0.000*	7.900	*0.063*	9.450	*0.001*
SGAIN	5.400	*0.219*	**4.200**	*0.043*	3.375	*0.696*	**3.350**	*0.181*	6.650	*0.262*	5.300	*1.000*
SGINI	6.950	*0.017*	4.550	*0.114*	4.875	*0.356*	3.600	*0.236*	2.400	*0.001*	6.150	*0.488*
WOE	**2.500**	*0.153*	7.250	*0.428*	**3.313**	*0.643*	6.100	*0.145*	7.100	*0.131*	9.450	*0.001*

Table 4.1 Hyper-heuristic using Number of Attributes Left and 5 heuristics compared to Standard Algorithms using Ranking Values of Single Data Set Experiments

Tables 4.1 and 4.2 show results for hyper-heuristics trained and tested on single data sets. The first table shows results obtained by hyper-heuristics using 5 heuristics (HH-5) while the second table shows results obtained by hyper-heuristics using all 12 heuristics (HH-12). Each column contains pairs of

	car		contrac		credit		derma		ecoli		flags	
HH-12	3.200	*p-value*	6.412	*p-value*	6.200	*p-value*	**3.750**	*p-value*	4.450	*p-value*	4.550	*p-value*
CHI	2.600	*0.443*	9.529	*0.007*	4.950	*0.247*	5.500	*0.088*	4.950	*0.649*	3.700	*0.374*
IG	**1.300**	*0.013*	6.294	*0.910*	6.000	*0.859*	4.050	*0.725*	**2.050**	*0.002*	5.100	*0.598*
GR	2.400	*0.312*	3.765	*0.036*	**4.200**	*0.061*	4.500	*0.421*	5.150	*0.537*	3.650	*0.354*
GINI	1.650	*0.046*	6.059	*0.741*	7.650	*0.221*	6.000	*0.023*	2.400	*0.009*	5.300	*0.468*
JM	11.050	*0.000*	5.294	*0.389*	5.850	*0.764*	6.450	*0.030*	3.500	*0.374*	6.250	*0.142*
MDL	2.400	*0.312*	6.412	*1.000*	6.250	*0.962*	6.700	*0.007*	6.500	*0.088*	3.150	*0.097*
MGINI	3.300	*0.909*	8.059	*0.191*	8.700	*0.026*	7.700	*0.000*	5.350	*0.382*	6.050	*0.173*
RLV	2.650	*0.555*	6.176	*0.834*	5.400	*0.450*	4.350	*0.525*	2.900	*0.054*	4.700	*0.879*
RLF	12.650	*0.000*	8.353	*0.172*	12.450	*0.000*	8.050	*0.001*	10.300	*0.000*	10.450	*0.000*
SGAIN	10.200	*0.000*	**3.471**	*0.007*	4.650	*0.085*	3.900	*0.872*	2.150	*0.010*	4.700	*0.876*
SGINI	2.850	*0.674*	5.471	*0.414*	4.450	*0.115*	5.100	*0.156*	5.000	*0.567*	**2.700**	*0.030*
WOE	11.950	*0.000*	7.765	*0.298*	6.300	*0.928*	7.150	*0.001*	7.400	*0.016*	7.500	*0.014*
	heart		ionosphere		spect		votes		wine		yeast	
HH-12	3.100	*p-value*	6.050	*p-value*	4.278	*p-value*	3.750	*p-value*	**3.400**	*p-value*	7.167	*p-value*
CHI	5.950	*0.006*	6.400	*0.764*	4.278	*1.000*	4.900	*0.398*	4.450	*0.268*	6.222	*0.511*
IG	7.300	*0.000*	5.750	*0.801*	5.056	*0.498*	4.250	*0.661*	4.450	*0.275*	4.056	*0.023*
GR	6.700	*0.001*	**4.050**	*0.059*	5.667	*0.282*	3.850	*0.931*	3.700	*0.738*	8.389	*0.417*
GINI	7.300	*0.000*	5.450	*0.620*	4.444	*0.886*	4.000	*0.819*	4.300	*0.334*	**3.722**	*0.012*
JM	5.800	*0.012*	6.950	*0.462*	5.611	*0.335*	4.350	*0.601*	4.500	*0.298*	5.944	*0.388*
MDL	5.250	*0.027*	4.300	*0.101*	4.167	*0.927*	4.150	*0.696*	3.850	*0.649*	7.222	*0.967*
MGINI	7.650	*0.000*	6.400	*0.755*	8.500	*0.005*	3.900	*0.888*	4.350	*0.314*	6.667	*0.734*
RLV	5.450	*0.015*	5.450	*0.641*	6.278	*0.147*	3.350	*0.681*	**3.400**	*1.000*	5.833	*0.369*
RLF	5.400	*0.062*	10.600	*0.000*	6.389	*0.131*	11.550	*0.000*	9.250	*0.000*	7.833	*0.655*
SGAIN	7.600	*0.000*	4.750	*0.247*	5.667	*0.323*	**2.350**	*0.120*	4.000	*0.543*	5.056	*0.164*
SGINI	7.650	*0.000*	5.100	*0.400*	4.444	*0.882*	2.950	*0.425*	4.100	*0.412*	5.222	*0.165*
WOE	**2.050**	*0.154*	5.650	*0.740*	**4.000**	*0.820*	5.400	*0.160*	8.350	*0.000*	9.000	*0.174*

Table 4.2 Hyper-heuristic using Number of Attributes Left and 12 heuristics compared to Standard Algorithms using Ranking Values of Single Data Set Experiments

values where the first value represents the mean rank of the method specified in the row header on the data set/s specified in the column header. We compare the various methods using ranking instead of classification accuracy because ranking is non-parameteric. The second value in each column is the resultant p-value when comparing the results of the method to the results of the hyper-heuristic on the same data set/s. The Student's t-test also uses mean rank to compare the two methods. We use the terms HH-5 and HH-12 to refer to the hyper-heuristic methods while for the rest of the methods we only specify the splitting heuristic used by the standard decision tree building algorithm, since this is the only thing that sets them apart. For these we use the same abbreviations detailed in 3.2.2.1.

In the single data set experiments, HH-5 did not manage to be the best overall performer on any of the 12 single data sets. HH-12 ranked first on both **derma** and **wine** however in neither of these cases is the difference between the hyper-heuristic and the second-best performing algorithm statistically significant. HH-5 managed to achieve an overall performance not significantly different (assuming a confidence level of 90%) to the best-performing method on 6 of the 12 data sets. HH-12 achieved this on 5 of the 12 data sets. The average predictive accuracy values for these results can be found in tables A.1 and A.2 in the Appendix.

4.1.4.2 Multiple Data Set Experiments

Tables 4.3 and 4.4 show results for HH-5 and HH-12 trained and tested on multiple data sets.

	ca, de, ec, wi		co, cr, io, sp		ye, vo, he, fl	
HH-5	3.850	*p-value*	5.353	*p-value*	**4.500**	*p-value*
CHI	5.900	*0.021*	5.412	*0.962*	7.200	*0.089*
IG	**3.350**	*0.556*	7.412	*0.072*	6.400	*0.247*
GR	5.250	*0.141*	7.059	*0.151*	5.400	*0.522*
GINI	6.200	*0.020*	7.353	*0.104*	10.100	*0.002*
JM	9.450	*0.000*	6.647	*0.262*	6.600	*0.237*
MDL	6.800	*0.002*	6.059	*0.529*	8.000	*0.052*
MGINI	7.050	*0.002*	10.529	*0.000*	9.700	*0.002*
RLV	4.250	*0.691*	6.882	*0.205*	5.000	*0.760*
RLF	12.550	*0.000*	10.059	*0.000*	9.900	*0.007*
SGAIN	7.750	*0.000*	**3.824**	*0.220*	5.000	*0.756*
SGINI	5.050	*0.154*	8.059	*0.017*	7.300	*0.112*
WOE	11.650	*0.000*	6.235	*0.464*	5.400	*0.626*

Table 4.3 Hyper-heuristic using Number of Attributes Left and 5 heuristics compared to Standard Algorithms using Ranking Values of Multiple Data Set Experiments

Each of the three columns shows results for experiments run on different groups of data sets:

ca, de, ec, wi: experiments using data sets **car**, **derma**, **ecoli** and **wine**.
co, cr, io, sp: experiments using data sets **contrac**, **credit**, **ionosphere** and **spect**.
ye, vo, he, fl: experiments using data sets **yeast**, **votes84**, **heart** and **flags**.

In each case, the rank value of each method is calculated using the average ranking obtained by the method over all four data sets over all the runs.

	ca, de, ec, wi		co, cr, io, sp		ye, vo, he, fl	
HH-12	4.900	*p-value*	4.857	*p-value*	**3.867**	*p-value*
CHI	4.300	*0.545*	5.762	*0.367*	5.867	*0.126*
IG	**3.600**	*0.185*	7.048	*0.060*	7.267	*0.015*
GR	6.750	*0.079*	6.381	*0.129*	6.000	*0.092*
GINI	5.450	*0.620*	8.143	*0.002*	9.467	*0.000*
JM	9.100	*0.000*	6.762	*0.075*	6.600	*0.027*
MDL	6.150	*0.215*	5.810	*0.373*	5.667	*0.150*
MGINI	7.050	*0.036*	9.905	*0.000*	10.067	*0.000*
RLV	4.500	*0.684*	7.429	*0.021*	7.200	*0.019*
RLF	12.800	*0.000*	12.000	*0.000*	10.533	*0.000*
SGAIN	7.900	*0.010*	**4.524**	*0.751*	6.733	*0.015*
SGINI	5.200	*0.770*	6.048	*0.237*	7.267	*0.014*
WOE	11.800	*0.000*	6.190	*0.175*	4.133	*0.824*

Table 4.4 Hyper-heuristic using Number of Attributes Left and 12 heuristics compared to Standard Algorithms using Ranking Values of Multiple Data Set Experiments

Though not statistically significant, both hyper-heuristics achieved the best overall ranking on the [**ye, vo, he, fl**] multiple data sets group. They were also never significantly worse than the top ranked method on the other data set groups. HH-5 managed to rank within the top three methods for all three data set groups. The average predictive accuracy values for all of these results can be found in tables A.3 and A.4 in the Appendix.

4.2 Hyper-heuristic Rules using Value Count of Attributes

4.2.1. Problem State Representation & Choice of Splitting Method

The second hyper-heuristic we present in this chapter uses the number of distinct values per candidate splitting attribute to characterize the problem state. We found various references in the literature to how this property can affect the performance of a splitting heuristic. For example, Quinlan devised the gain ratio heuristic as an alternative to the information gain heuristic as the latter suffers from a bias that favours attributes with many values (Quinlan, 1993). Mántaras (1991) proposed using symmetric gain ratio for the same exact reason. Zhou and Dillon (1991) extended the Gini index to create the symmetric Gini index because they maintained that the former, like information gain, has a bias favouring attributes with many values. It seems that the performance of some splitting heuristics can be effected by the number of values each attribute can take. We therefore represent a hyper-heuristic that decides on how to split the data when faced with a data partition $\boldsymbol{d}$ as a set of $\boldsymbol{m}$ rules:

```
IF          $x_1\% < low\_value\_count_1$ AND
            $y_1\% > high\_value\_count_1$ THEN use heuristic $h_1$
ELSE IF     $x_2\% < low\_value\_count_2$ AND
            $y_2\% > high\_value\_count_2$ THEN use heuristic $h_2$
...
ELSE IF     $x_{m-1}\% < low\_value\_count_{m-1}$ AND
            $y_{m-1}\% > high\_value\_count_{m-1}$ THEN use heuristic $h_{m-1}$
ELSE        use heuristic $h_m$
```

where, $\boldsymbol{x_i}$ and $\boldsymbol{y_i}$ are both percentage values ranging from 0 to 100,

low_ value_ count$_i$ and ***high_ value_ count***$_i$ are thresholds for value count,

h$_i$ is the heuristic to use to sort the list of candidate splitting attributes should rule ***i*** be triggered.

As in the earlier hyper-heuristics, each rule is effectively asking a question about the current problem state where the problem state is the partition that needs to be split. In this case, the hyper-heuristic is looking at the set of candidate splitting attributes remaining in the partition to be split and asking a question related to the number of distinct values each attribute has. More specifically, the question being asked by each hyper-heuristic rule is: "*Do x% of the attributes have a low value count, (100-(x+y))% of the attributes have a medium value count and y% of the attributes have a high value count?*".

What do we mean by low, medium and high value count? A low value count range is defined by an integer upper bound value ***low_ value_ count*** such that any attribute that takes ***a*** distinct values where ***a*** < ***low_ value_ count*** is said to be an attribute with a low value count. A medium value count range is defined by two integer values ***low_ value_ count*** and ***high_ value_ count*** such that that any attribute that takes ***b*** distinct values where ***low_ value_ count*** ≤ ***b*** ≤ ***high_ value_ count*** is said to be an attribute with a medium value count. A high value count range is defined by an integer lower bound value ***high_ value_ count*** such that any attribute that takes ***c*** distinct values where ***c*** > ***high_ value_ count*** is said to be an attribute with a high value count. Within a hyper-heuristic rule set, each rule defines its own ranges for low, medium and high value count with the condition that these ranges cannot overlap – this means that each rule has its own definition for ***low_ value_ count*** and ***high_ value_ count*** while respecting the condition ***low_ value_ count*** ≤ ***high_ value_ count***.

```
Let D = our initial data partition /*the whole training set*/
Let A = list of all attributes in D
Let T = <empty tree>
Let n = <empty tree node>

Insert n in T
CALL GrowTree(D, A, n)

GrowTree(D, A, n)
    IF <all instances in D are of the same class> THEN
        c = class of instances in D
        Create leaf at n with label c
    ELSE
        VC = <empty set> /*VC will contain value count of each attribute*/
        FOR EACH a in A
            vc = number of distinct values a can take
            Insert vc in VC
        END /*FOR EACH*/
        h = GetHeuristic(VC)
        Sort A using heuristic h
        a = attribute ranked 1st in A /*the splitting attribute*/
        Use attribute a for node n
        V = set of all distinct values a can have
        FOR EACH v in V
            Create a branch b from n using value v
            Create a new empty child node nv at branch b
            Dv = data partition containing only instances where a = v
            Av = A minus a
            GrowTree(Dv, Av, nv)
        END /*FOR EACH*/
    END /*IF*/
END /*GrowTree*/

GetHeuristic(VC)
    Use hyper-heuristic rules to return heuristic h based on VC
END /*GetHeuristic*/
```

Figure 4.2 Hyper-heuristic Decision Tree Building Algorithm that Chooses Heuristic using Value Count of Attributes

Using this set of hyper-heuritic rules, whenever a data partition needs to be split, we look at the number of distinct values each candidate splitting attribute can take. We then go through each of the rule conditionals one by one to see if the problem state described by any one of the conditionals matches the current state of the problem we are solving. If one of the rule conditionals fits the description of the current partition we want to split, we use the heuristic dictated by that rule to sort the candidate splitting attributes. If none of the conditionals of the first $\boldsymbol{m}$ - 1 rules test positive to the problem state, the default heuristic $\boldsymbol{h}_m$ is used. We present the pseudo-code for the decision-tree building algorithm that uses this hyper-heuristic in Figure 4.2.

We again ran two sets of experiments using this hyper-heuristic: one set uses all 12 heuristics while the other set uses the same reduced set of 5 heuristics described in 4.1.1.

4.2.2 Genetic Algorithm

We used the same genetic algorithm described in 4.1.2 to search for good hyper-heuristic rules that use the value count of attributes. The only difference between this genetic algorithm and the previous one is in the encoding of the individuals. In this case, for each hyper-heuristic rule we need to define the heuristic to be used, three ranges for low, medium and high value count as well as three percentage values $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ to represent the percentage of attributes that have a low, medium and high value count.

To define three ranges for low, medium and high value count it suffices to specify two integer values $\boldsymbol{l}$ and $\boldsymbol{u}$ such that all attributes that have a value count smaller than $\boldsymbol{l}$ are considered to have a low value count, all attributes that have a value count bigger than $\boldsymbol{u}$ are considered to have a high value count while all other attributes are considered to have a medium value count. This means that any attribute is guaranteed to fall into anyone of these three categories. Thus, an individual in the genetic algorithm encoded our hyper-heuristic in the following manner:

$$\boldsymbol{l_1, x_1, u_1, y_1, h_1, l_2, x_2, u_2, y_2, h_2, \ldots, l_{m-1}, x_{m-1}, u_{m-1}, y_{m-1}, h_{m-1}, h_m}$$

where for any rule i:

- $\boldsymbol{l_i}$ and $\boldsymbol{u_i}$ are integer values defining three ranges for low, medium and high value count,
- $\boldsymbol{x_i}$ and $\boldsymbol{y_i}$ are integer values representing the percentage of attributes that have a low value count and high value count respectively,
- $\boldsymbol{h_i}$ is an integer value indexing the heuristic to be used to split the data should rule $\boldsymbol{i}$ be triggered.

In this way, we encode each hyper-heuristic rule using just 5 integer values. Whenever the genetic algorithm created or manipulated the genes of an individual, we made sure that these conditions were respected:

- $2 \leq \boldsymbol{u_i} \leq 6$,
- $\boldsymbol{l_i} \leq \boldsymbol{u_i}$,
- $\boldsymbol{x_i}, \boldsymbol{y_i} > 0$,
- $(\boldsymbol{x_i} + \boldsymbol{y_i}) \leq 100$.

In order to facilitate the search carried out by the genetic algorithm, we also decided to restrict the number of values each percentage gene (i.e. $\boldsymbol{x_i}$ and $\boldsymbol{y_i}$) can take. More specifically, these percentage genes were only allowed to take one of eleven possible values from the set $\{0, 10, 20, \ldots, 100\}$. This reduces the search space of the genetic algorithm thus making it easier for it to converge to an effective set of hyper-heuristic rules.

For this hyper-heuristic we used rule sets of size 4 where the 4th rule is the default heuristic. The rest of the parameters of the genetic algorithm were kept the same as the ones used in the experiments described in 4.1. We also utilized the same data sets in the same exact manner. Each experiment was run 20-40 times, each time varying the hyper-heuristic training set and test set as well as the random seed of the initial population of the genetic algorithm.

4.2.3 Results

4.2.3.1 Single Data Set Experiments

	car		contrac		credit		derma		ecoli		flags	
HH-5	2.950	*p-value*	5.300	*p-value*	3.850	*p-value*	3.950	*p-value*	5.950	*p-value*	5.300	*p-value*
CHI	2.350	*0.439*	7.450	*0.055*	5.600	*0.102*	5.250	*0.224*	3.200	*0.009*	4.900	*0.725*
IG	**1.400**	*0.028*	6.400	*0.284*	7.700	*0.000*	4.800	*0.444*	2.300	*0.001*	4.350	*0.373*
GR	2.200	*0.340*	**5.050**	*0.774*	**3.800**	*0.955*	4.700	*0.494*	6.900	*0.463*	**3.600**	*0.165*
GINI	1.500	*0.045*	5.950	*0.509*	8.750	*0.000*	4.850	*0.440*	2.150	*0.000*	4.050	*0.230*
JM	11.000	*0.000*	5.150	*0.891*	5.950	*0.038*	5.500	*0.186*	6.700	*0.578*	6.400	*0.403*
MDL	2.100	*0.255*	6.000	*0.540*	6.300	*0.034*	6.000	*0.071*	5.700	*0.838*	6.000	*0.594*
MGINI	2.650	*0.710*	9.100	*0.001*	8.400	*0.000*	5.600	*0.179*	3.200	*0.012*	5.950	*0.614*
RLV	3.450	*0.602*	6.750	*0.139*	6.200	*0.027*	**3.400**	*0.584*	**2.050**	*0.000*	4.150	*0.341*
RLF	12.400	*0.000*	9.450	*0.001*	11.550	*0.000*	9.950	*0.000*	9.350	*0.006*	7.800	*0.081*
SGAIN	10.050	*0.000*	5.500	*0.850*	4.200	*0.689*	3.700	*0.825*	4.500	*0.215*	4.750	*0.628*
SGINI	2.100	*0.251*	5.250	*0.961*	4.600	*0.486*	4.250	*0.779*	4.250	*0.124*	5.050	*0.838*
WOE	12.400	*0.000*	6.400	*0.319*	5.750	*0.116*	7.850	*0.002*	7.800	*0.179*	8.800	*0.006*

	heart		ionosphere		spect		votes		wine		yeast	
HH-5	**3.000**	*p-value*	5.200	*p-value*	5.100	*p-value*	6.250	*p-value*	5.000	*p-value*	6.700	*p-value*
CHI	7.200	*0.000*	6.650	*0.166*	6.400	*0.509*	4.900	*0.296*	4.200	*0.438*	**5.300**	*0.220*
IG	5.050	*0.015*	5.850	*0.546*	5.100	*1.000*	4.400	*0.115*	4.250	*0.448*	6.300	*0.717*
GR	7.350	*0.000*	4.150	*0.311*	6.800	*0.405*	3.950	*0.055*	3.750	*0.196*	5.800	*0.376*
GINI	7.550	*0.000*	4.950	*0.828*	5.400	*0.867*	4.400	*0.112*	5.600	*0.557*	5.800	*0.375*
JM	4.850	*0.053*	8.200	*0.016*	**3.400**	*0.360*	5.850	*0.755*	3.450	*0.163*	7.500	*0.519*
MDL	6.500	*0.000*	**3.900**	*0.170*	6.000	*0.652*	3.150	*0.004*	**3.250**	*0.082*	7.150	*0.678*
MGINI	8.050	*0.000*	4.800	*0.720*	8.000	*0.132*	3.450	*0.015*	6.100	*0.382*	6.100	*0.581*
RLV	6.300	*0.002*	5.150	*0.967*	4.600	*0.781*	3.700	*0.036*	4.600	*0.696*	6.750	*0.967*
RLF	4.650	*0.114*	10.150	*0.000*	5.000	*0.960*	11.450	*0.000*	8.900	*0.007*	6.750	*0.969*
SGAIN	6.400	*0.001*	**3.900**	*0.200*	5.000	*0.961*	4.300	*0.110*	5.900	*0.452*	5.700	*0.414*
SGINI	7.000	*0.000*	4.550	*0.532*	6.200	*0.561*	**2.900**	*0.002*	5.500	*0.645*	**5.300**	*0.253*
WOE	3.100	*0.908*	5.350	*0.902*	3.800	*0.506*	5.450	*0.533*	6.350	*0.267*	6.850	*0.903*

Table 4.5 Hyper-heuristic using Attribute Value Count and 5 heuristics compared to Standard Algorithms using Ranking Values of Single Data Set Experiments

Tables 4.5 and 4.6 show results for HH-5 and HH-12 trained and tested on single data sets. Though not statistically significant, both HH-5 and HH-12 managed to rank first on the **hearts** data set. The overall performance of HH-5 on 8 of the 12 data sets was not significantly different from that of the best-performing method in each case. HH-12 achieved managed this on 9 of the 12 data sets. The average predictive accuracy values for all of these results can be found in tables A.5 and A.6 in the Appendix.

	car		contrac		credit		derma		ecoli		flags	
HH-12	1.950	*p-value*	6.059	*p-value*	6.400	*p-value*	5.750	*p-value*	5.950	*p-value*	4.850	*p-value*
CHI	2.550	*0.355*	6.235	*0.905*	5.650	*0.439*	5.200	*0.664*	**3.500**	*0.024*	4.650	*0.844*
IG	**1.400**	*0.290*	7.941	*0.169*	7.050	*0.509*	4.100	*0.154*	4.250	*0.110*	5.100	*0.825*
GR	2.150	*0.742*	5.647	*0.775*	6.150	*0.820*	5.650	*0.938*	5.650	*0.805*	4.150	*0.523*
GINI	1.800	*0.798*	7.176	*0.456*	8.100	*0.066*	6.300	*0.660*	4.400	*0.123*	5.800	*0.330*
JM	10.900	*0.000*	6.412	*0.810*	3.600	*0.010*	5.850	*0.932*	4.500	*0.229*	6.350	*0.182*
MDL	2.000	*0.935*	6.353	*0.845*	6.200	*0.853*	6.150	*0.751*	4.600	*0.261*	4.600	*0.805*
MGINI	3.050	*0.157*	7.941	*0.207*	7.700	*0.224*	7.750	*0.132*	4.850	*0.317*	7.350	*0.022*
RLV	3.350	*0.133*	7.059	*0.436*	5.450	*0.392*	**3.900**	*0.110*	**3.500**	*0.021*	4.850	*1.000*
RLF	12.700	*0.000*	7.118	*0.532*	12.050	*0.000*	8.600	*0.059*	8.750	*0.054*	9.500	*0.000*
SGAIN	10.250	*0.000*	**4.353**	*0.214*	**3.200**	*0.001*	4.750	*0.476*	4.000	*0.076*	4.950	*0.917*
SGINI	2.550	*0.355*	5.471	*0.660*	6.350	*0.964*	4.950	*0.502*	5.700	*0.829*	**3.300**	*0.059*
WOE	11.850	*0.000*	6.471	*0.776*	4.550	*0.074*	7.200	*0.243*	6.200	*0.853*	8.050	*0.009*
	heart		ionosphere		spect		votes		wine		yeast	
HH-12	**1.950**	*p-value*	4.550	*p-value*	4.818	*p-value*	3.650	*p-value*	3.900	*p-value*	4.950	*p-value*
CHI	6.500	*0.000*	5.700	*0.307*	6.364	*0.288*	4.000	*0.781*	4.350	*0.659*	6.750	*0.106*
IG	5.850	*0.000*	5.350	*0.492*	5.909	*0.450*	4.650	*0.369*	4.150	*0.799*	5.500	*0.589*
GR	6.850	*0.000*	5.400	*0.424*	8.000	*0.094*	3.000	*0.525*	3.000	*0.385*	7.000	*0.087*
GINI	8.500	*0.000*	**3.850**	*0.454*	6.455	*0.287*	4.950	*0.232*	4.100	*0.842*	**4.700**	*0.788*
JM	6.900	*0.000*	7.800	*0.004*	6.091	*0.402*	4.150	*0.660*	**2.500**	*0.184*	6.850	*0.099*
MDL	6.350	*0.000*	4.900	*0.736*	4.182	*0.659*	3.350	*0.780*	4.400	*0.627*	7.250	*0.060*
MGINI	7.950	*0.000*	7.500	*0.022*	7.818	*0.084*	4.550	*0.425*	6.350	*0.032*	6.300	*0.218*
RLV	6.250	*0.000*	4.750	*0.853*	**3.545**	*0.337*	4.250	*0.587*	4.150	*0.836*	**4.700**	*0.826*
RLF	5.300	*0.005*	9.000	*0.001*	4.273	*0.728*	10.850	*0.000*	11.000	*0.000*	8.650	*0.004*
SGAIN	7.550	*0.000*	4.500	*0.958*	3.909	*0.542*	3.200	*0.663*	4.650	*0.549*	7.300	*0.059*
SGINI	7.400	*0.000*	6.150	*0.134*	6.636	*0.226*	**2.900**	*0.416*	2.900	*0.284*	6.050	*0.281*
WOE	2.700	*0.196*	5.900	*0.236*	3.818	*0.467*	6.750	*0.027*	4.650	*0.580*	6.000	*0.409*

Table 4.6 Hyper-heuristic using Attribute Value Count and 12 heuristics compared to Standard Algorithms using Ranking Values of Single Data Set Experiments

4.2.3.2 Multiple Data Set Experiments

Tables 4.7 and 4.8 show results for HH-5 and HH-12 trained and tested on multiple data sets.

	ca, de, ec, wi		co, cr, io, sp		ye, vo, he, fl	
HH-5	5.150	*p-value*	6.000	*p-value*	5.179	*p-value*
CHI	**4.500**	*0.339*	5.864	*0.903*	6.786	*0.121*
IG	**4.500**	*0.316*	8.636	*0.011*	6.964	*0.069*
GR	5.575	*0.552*	**4.591**	*0.206*	6.250	*0.292*
GINI	5.350	*0.779*	6.500	*0.638*	7.464	*0.024*
JM	9.150	*0.000*	6.727	*0.516*	6.821	*0.116*
MDL	6.725	*0.030*	6.545	*0.604*	6.929	*0.092*
MGINI	6.425	*0.064*	10.545	*0.000*	9.179	*0.000*
RLV	4.925	*0.733*	6.091	*0.934*	5.857	*0.497*
RLF	12.450	*0.000*	12.227	*0.000*	9.536	*0.000*
SGAIN	7.650	*0.001*	4.955	*0.352*	8.071	*0.005*
SGINI	5.000	*0.822*	7.045	*0.334*	7.464	*0.019*
WOE	11.400	*0.000*	5.136	*0.432*	**4.214**	*0.361*

Table 4.7 Hyper-heuristic using Attribute Value Count and 5 heuristics compared to Standard Algorithms using Ranking Values of Multiple Data Set Experiments

	ca, de, ec, wi		co, cr, io, sp		ye, vo, he, fl	
HH-12	4.725	*p-value*	6.462	*p-value*	**3.314**	*p-value*
CHI	4.425	*0.679*	6.615	*0.879*	7.086	*0.000*
IG	4.800	*0.918*	6.769	*0.750*	7.114	*0.000*
GR	**4.175**	*0.445*	5.654	*0.417*	6.771	*0.000*
GINI	5.525	*0.234*	8.077	*0.073*	7.714	*0.000*
JM	8.775	*0.000*	7.000	*0.601*	6.429	*0.000*
MDL	6.525	*0.016*	6.346	*0.913*	7.143	*0.000*
MGINI	7.400	*0.000*	9.808	*0.001*	9.743	*0.000*
RLV	5.075	*0.611*	6.038	*0.688*	6.114	*0.001*
RLF	12.625	*0.000*	11.423	*0.000*	9.686	*0.000*
SGAIN	8.450	*0.000*	**4.192**	*0.017*	6.829	*0.000*
SGINI	5.350	*0.391*	6.885	*0.677*	7.200	*0.000*
WOE	11.500	*0.000*	5.654	*0.423*	5.686	*0.007*

Table 4.8 Hyper-heuristic using Attribute Value Count and 12 heuristics compared to Standard Algorithms using Ranking Values of Multiple Data Set Experiments

HH-12 managed an average ranking significantly better than all the standard algorithms on the [**ye, vo, he, fl**] multiple data sets group. HH-5 did not manage to come first in this group of data sets though its performance is not significantly different from the best performing method (WOE). As regards to the other two groups of data sets, HH-5 was never significantly worse than the best performing method of each group. This is contrast to HH-12 whose performance on [**co, cr, io, sp**] was significantly worse than that of the best performing method. The average predictive accuracy values for all of these results can be found in tables A.7 – A.8 in the Appendix.

4.3 Hyper-heuristic Rules using Attribute Entropy

4.3.1. Problem State Representation & Choice of Splitting Method

We have already discussed how the number of distinct values a splitting attribute can take affects the way it is ranked by certain splitting heuristics. This is due to the inherent bias found in some heuristics favouring attributes with many values, irrespective of whether choosing these attributes to construct tree nodes will actually produce accurate decision trees. We have seen how using information about the value count of splitting attributes can go some way in helping decide which heuristic to use to split the data while building a decision tree. This section presents an alternative solution to the same problem by proposing hyper-heuristic rules that use the entropy of each splitting attribute to characterize the problem state.

Entropy (see Equation 3.1) is affected by various statistical features of an attribute. The higher the number of distinct values an attribute can take, the higher its entropy value will be. Also, the more balanced these values are in the data partition, the higher the entropy of that attribute. An attribute in a data partition is said to have perfectly balanced values if for each value there are the same number of instances in that data partition with that value. Since the entropy value is a reflection of more than one statistical feature of an attribute, one can say that a problem state description that uses entropy to characterize a data partition is a richer description than one that uses value count.

We therefore represent a hyper-heuristic that decides on how to split the data when faced with a data partition $\boldsymbol{d}$ as a set of $\boldsymbol{m}$ rules:

```
IF          $x_1 < low\_entropy_1$ AND
            $y_1 > high\_entropy_1$ THEN use heuristic $h_1$
ELSE IF     $x_2 < low\_entropy_2$ AND
            $y_2 > high\_entropy_2$ THEN use heuristic $h_2$
...
ELSE IF     $x_{m-1} < low\_entropy_{m-1}$ AND
            $y_{m-1} > high\_entropy_{m-1}$ THEN use heuristic $h_{m-1}$
ELSE use heuristic $h_m$
```

where, $\boldsymbol{x_i}$ and $\boldsymbol{y_i}$ are both percentage values ranging from 0 to 100,

$\boldsymbol{low\_entropy_i}$ and $\boldsymbol{high\_entropy_i}$ are thresholds for entropy,

$\boldsymbol{h_i}$ is the heuristic to use to sort the list of candidate splitting attributes should rule $\boldsymbol{i}$ be triggered.

As in the previous hyper-heuristic, when faced with a data partition each rule tries to insert each candidate splitting attribute into one of three bins. In this case, the three bins represent attributes with low entropy, medium entropy and high entropy. A low entropy bin is defined by a range that is specified by an upper bound real value $\boldsymbol{low\_entropy}$ such that any attribute that has entropy $\boldsymbol{a}$ where $\boldsymbol{a} < \boldsymbol{low\_entropy}$ is said to be an attribute with low entropy. A medium entropy bin is defined by a range that is specified by two real values ***low_ entropy*** and ***high_ entropy*** such that that any attribute of entropy ***b*** where ***low_ entropy*** $\leq$ ***b*** $\leq$ ***high_ entropy*** is said to be an attribute with medium entropy.

```
Let D = our initial data partition /*the whole training set*/
Let A = list of all attributes in D
Let T = <empty tree>
Let n = <empty tree node>

Insert n in T
CALL GrowTree(D, A, n)

GrowTree(D, A, n)
    IF <all instances in D are of the same class> THEN
        c = class of instances in D
        Create leaf at n with label c
    ELSE
        E = <empty set> /*E will contain entropy of each attribute*/
        FOR EACH a in A
            e = entropy of a
            Insert e in E
        END /*FOR EACH*/
        h = GetHeuristic(E)
        Sort A using heuristic h
        a = attribute ranked 1st in A /*the splitting attribute*/
        Use attribute a for node n
        V = set of all distinct values a can have
        FOR EACH v in V
            Create a branch b from n using value v
            Create a new empty child node nv at branch b
            Dv = data partition containing only instances where a = v
            Av = A minus a
            CALL GrowTree(Dv, Av, nv)
        END /*FOR EACH*/
    END /*IF*/
END /*GrowTree*/

GetHeuristic(E)
    Use hyper-heuristic rules to return heuristic h based on E
END /*GetHeuristic*/
```

Figure 4.3 Hyper-heuristic Decision Tree Building Algorithm that Chooses Heuristic using Entropy of Attributes

A high entropy bin is defined by a range that is specifed by a lower bound real value ***high_ entropy*** such that any attribute that has entropy ***c*** where $\boldsymbol{c} >$ ***high_ entropy*** is said to be an attribute with high entropy. Each rule defines its own ranges for low, medium and high entropy with the condition that these ranges cannot overlap within the same rule – this means that each rule has its own definition for ***low_ entropy*** and ***high_ entropy*** while respecting the condition ***low_ entropy*** ≤ ***high_ entropy***.

Given a data partition that needs to be split, the hyper-heuristic calculates the entropy of each attribute and tries to find a rule whose conditional matches the make-up of the current problem state. If a match is found, the splitting heuristic of the matching rule is used to split the data; otherwise the default heuristic $\boldsymbol{h_m}$ is used. The pseudo-code for the decision-tree building algorithm that uses this hyper-heuristic is displayed in Figure 4.3.

4.3.2 Genetic Algorithm

We use the same genetic algorithm described in 4.2.2 with the only difference being in the way each hyper-heuristic rule-set is encoded. Each hyper-heuristic rule set was encoded as a combination of integer and real values:

$$\boldsymbol{l_1, x_1, u_1, y_1, h_1, \; l_2, x_2, u_2, y_2, h_2, \ldots, l_{m-1}, x_{m-1}, u_{m-1}, y_{m-1}, \; h_{m-1}, h_m}$$

where for any rule *i*:

- $\boldsymbol{l_i}$ and $\boldsymbol{u_i}$ are real values defining three ranges for low, medium and high entropy,
- $\boldsymbol{x_i}$ and $\boldsymbol{y_i}$ are integer values representing the percentage of attributes that have low entropy and high entropy respectively,

- $\boldsymbol{h_i}$ is an integer value indexing the heuristic to be used to split the data should rule $\boldsymbol{i}$ be triggered.

We used the same granularity for the percentage genes as in the previous experiments. This means that each percentage gene can take one of eleven possible values from the set {0, 10, 20, ..., 100} while respecting the condition $(\boldsymbol{x_i} + \boldsymbol{y_i}) \leq 100$. We programmed the threshold genes $\boldsymbol{l_i}$ that represent the low entropy range to take one of eleven possible values from the set {0.001, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5}. The genes $\boldsymbol{u_i}$ that define the high entropy range were programmed to take one value from the set {1, 1.34, 1.68, 2.02, 2.36, 2.7, 3.04, 3.38, 3.72, 4.06, 4.4}. We chose these values for the threshold genes after analyzing frequency distribution graphs of the entropy values of the attributes in our data sets. The values used to make up these graphs were produced by running standard ID3 on these data sets while storing the entropy of each candidate splitting attribute at each point of the decision-tree building process.

As before, we ran single and multiple data set experiments where each single data set experiment was run 20 times and each multiple data set experiments was run 100 times, each time varying the hyper-heuristic training set and test set as well as the random seed of the initial population of the genetic algorithm. The results of this chapter were published in Vella et al (2009).

4.3.3 Results

4.3.3.1 Single Data Set Experiments

	car		contrac		credit		derma		ecoli		flags	
HH-5	1.900	*p-value*	5.533	*p-value*	4.750	*p-value*	5.450	*p-value*	4.450	*p-value*	5.450	*p-value*
CHI	2.300	*0.466*	6.400	*0.535*	5.300	*0.613*	5.150	*0.790*	**3.150**	*0.250*	3.950	*0.184*
IG	**1.450**	*0.318*	7.533	*0.166*	8.100	*0.001*	**3.800**	*0.126*	3.900	*0.572*	4.250	*0.271*
GR	2.300	*0.466*	**4.867**	*0.613*	5.550	*0.423*	3.900	*0.190*	3.750	*0.520*	5.150	*0.771*
GINI	**1.450**	*0.318*	7.667	*0.147*	7.800	*0.004*	5.100	*0.761*	4.000	*0.640*	4.600	*0.452*
JM	11.150	*0.000*	6.800	*0.439*	4.950	*0.848*	5.300	*0.910*	5.500	*0.436*	4.800	*0.545*
MDL	2.300	*0.466*	5.200	*0.811*	5.400	*0.535*	6.300	*0.496*	5.050	*0.634*	5.050	*0.693*
MGINI	3.100	*0.092*	9.067	*0.015*	8.700	*0.000*	7.350	*0.146*	5.650	*0.294*	6.800	*0.213*
RLV	3.050	*0.174*	6.533	*0.455*	5.800	*0.323*	4.150	*0.262*	3.700	*0.448*	5.650	*0.859*
RLF	12.700	*0.000*	8.867	*0.027*	12.300	*0.000*	9.450	*0.002*	9.250	*0.001*	10.600	*0.000*
SGAIN	10.150	*0.000*	4.933	*0.653*	**3.900**	*0.376*	4.200	*0.323*	3.850	*0.586*	4.250	*0.277*
SGINI	2.700	*0.214*	5.733	*0.876*	6.250	*0.176*	5.950	*0.695*	4.300	*0.892*	**3.050**	*0.030*
WOE	11.850	*0.000*	5.600	*0.960*	4.800	*0.964*	6.750	*0.325*	6.500	*0.128*	7.200	*0.175*

	heart		ionosphere		spect		votes		wine		yeast	
HH-5	3.100	*p-value*	4.650	*p-value*	4.538	*p-value*	3.400	*p-value*	4.100	*p-value*	5.727	*p-value*
CHI	6.450	*0.001*	6.300	*0.121*	7.231	*0.030*	4.150	*0.467*	4.300	*0.864*	6.818	*0.512*
IG	6.150	*0.001*	5.150	*0.646*	6.231	*0.176*	5.500	*0.010*	3.400	*0.499*	6.091	*0.807*
GR	7.650	*0.000*	**3.850**	*0.348*	6.000	*0.313*	**3.100**	*0.702*	**3.300**	*0.450*	7.818	*0.250*
GINI	8.200	*0.000*	5.600	*0.343*	7.000	*0.039*	5.000	*0.045*	4.550	*0.712*	7.091	*0.390*
JM	4.200	*0.240*	6.950	*0.021*	**3.154**	*0.199*	5.400	*0.031*	4.850	*0.575*	7.455	*0.338*
MDL	6.700	*0.001*	4.500	*0.856*	4.462	*0.948*	5.300	*0.075*	4.000	*0.931*	6.364	*0.694*
MGINI	9.350	*0.000*	6.850	*0.072*	7.538	*0.035*	5.450	*0.022*	7.050	*0.022*	7.182	*0.366*
RLV	4.750	*0.026*	5.400	*0.503*	4.846	*0.780*	5.300	*0.026*	3.350	*0.498*	**4.727**	*0.455*
RLF	4.000	*0.350*	9.900	*0.000*	3.769	*0.505*	8.800	*0.000*	9.700	*0.000*	6.455	*0.657*
SGAIN	7.800	*0.000*	**3.850**	*0.370*	4.538	*1.000*	3.150	*0.686*	4.900	*0.516*	**4.727**	*0.520*
SGINI	7.250	*0.000*	5.700	*0.325*	7.077	*0.035*	4.250	*0.287*	3.350	*0.471*	5.273	*0.764*
WOE	**1.550**	*0.026*	5.550	*0.389*	3.231	*0.312*	5.050	*0.131*	4.400	*0.797*	6.182	*0.767*

Table 4.9 Hyper-heuristic using Attribute Entropy and 5 heuristics compared to Standard Algorithms using Ranking Values of Single Data Set Experiments

Tables 4.9 and 4.10 show results for HH-5 and HH-12 trained and tested on single data sets. As in the experiments presented in 4.2, HH-12 ranked first on the **hearts** data set, though the difference in average ranking is not statistically signficant from the second best result. HH-5 did not manage to top any of the results tables for these experiments. However, for 10 of the 12 data sets, there was no significant difference between the performance of HH-5 and the best performing method of each data set. The same can be said for HH-12 for 8 of the 12 data sets. The average predictive accuracy values for all of these results can be found in tables A.9 and A.10 in the Appendix.

	car		contrac		credit		derma		ecoli		flags	
HH-12	3.900	*p-value*	5.000	*p-value*	5.000	*p-value*	3.350	*p-value*	3.450	*p-value*	6.550	*p-value*
CHI	2.500	*0.111*	6.500	*0.252*	4.600	*0.684*	4.250	*0.372*	5.100	*0.163*	5.200	*0.283*
IG	**1.250**	*0.002*	6.944	*0.097*	7.550	*0.022*	4.250	*0.340*	3.250	*0.836*	3.850	*0.011*
GR	1.950	*0.022*	**4.444**	*0.663*	5.150	*0.900*	5.750	*0.060*	6.500	*0.012*	**3.350**	*0.004*
GINI	1.600	*0.008*	7.444	*0.061*	8.300	*0.007*	5.100	*0.080*	3.000	*0.628*	4.350	*0.033*
JM	10.550	*0.000*	5.833	*0.490*	4.900	*0.931*	6.750	*0.001*	6.250	*0.020*	6.250	*0.806*
MDL	2.200	*0.052*	7.500	*0.077*	6.850	*0.095*	5.300	*0.039*	5.200	*0.147*	4.900	*0.124*
MGINI	3.350	*0.580*	8.556	*0.006*	8.500	*0.006*	7.250	*0.001*	3.050	*0.673*	6.450	*0.930*
RLV	2.300	*0.102*	6.222	*0.313*	6.100	*0.327*	4.450	*0.290*	**2.800**	*0.468*	4.950	*0.095*
RLF	12.750	*0.000*	8.778	*0.003*	11.800	*0.000*	9.450	*0.000*	9.200	*0.000*	9.000	*0.055*
SGAIN	10.200	*0.000*	4.556	*0.727*	**3.850**	*0.317*	**3.300**	*0.959*	3.700	*0.820*	4.900	*0.104*
SGINI	2.350	*0.085*	7.389	*0.039*	4.650	*0.742*	4.750	*0.102*	4.150	*0.488*	4.300	*0.056*
WOE	12.200	*0.000*	5.889	*0.487*	4.850	*0.888*	8.500	*0.000*	7.300	*0.006*	9.900	*0.007*
	heart		ionosphere		spect		votes		wine		yeast	
HH-12	**1.550**	*p-value*	4.350	*p-value*	5.000	*p-value*	5.650	*p-value*	4.550	*p-value*	7.455	*p-value*
CHI	7.100	*0.000*	4.000	*0.682*	6.000	*0.374*	6.150	*0.706*	3.850	*0.507*	8.091	*0.687*
IG	6.050	*0.000*	6.000	*0.099*	4.714	*0.822*	4.450	*0.347*	3.700	*0.410*	6.818	*0.697*
GR	6.950	*0.000*	4.600	*0.757*	5.500	*0.741*	**2.350**	*0.003*	3.900	*0.561*	6.818	*0.692*
GINI	7.750	*0.000*	6.150	*0.080*	6.214	*0.314*	4.050	*0.167*	**3.000**	*0.101*	5.000	*0.121*
JM	6.250	*0.000*	6.800	*0.022*	5.286	*0.835*	5.450	*0.875*	4.000	*0.629*	4.909	*0.121*
MDL	7.250	*0.000*	4.200	*0.857*	7.786	*0.045*	3.700	*0.111*	4.150	*0.703*	7.273	*0.906*
MGINI	9.850	*0.000*	6.450	*0.055*	7.857	*0.026*	4.500	*0.323*	6.100	*0.175*	**4.455**	*0.062*
RLV	5.100	*0.000*	**3.900**	*0.590*	5.643	*0.644*	5.800	*0.902*	6.450	*0.121*	5.909	*0.300*
RLF	4.450	*0.001*	9.900	*0.000*	5.643	*0.663*	10.800	*0.000*	8.100	*0.015*	6.182	*0.515*
SGAIN	6.200	*0.000*	5.350	*0.291*	3.857	*0.370*	3.100	*0.024*	4.500	*0.962*	7.455	*1.000*
SGINI	7.700	*0.000*	5.350	*0.207*	6.643	*0.201*	2.950	*0.017*	3.200	*0.165*	4.727	*0.098*
WOE	2.750	*0.050*	6.450	*0.047*	**3.786**	*0.328*	6.650	*0.437*	6.700	*0.094*	7.455	*1.000*

Table 4.10 Hyper-heuristic using Attribute Entropy and 12 heuristics compared to Standard Algorithms using Ranking Values of Single Data Set Experiments

4.3.3.2 Multiple Data Set Experiments

Tables 4.11 and 4.12 show results for HH-5 and HH-12 trained and tested on multiple data sets.

	ca, de, ec, wi		co, cr, io, sp		ye, vo, he, fl	
HH-5	5.070	*p-value*	5.680	*p-value*	**4.530**	*p-value*
CHI	4.730	*0.409*	7.210	*0.002*	6.850	*0.000*
IG	4.670	*0.330*	6.950	*0.007*	6.460	*0.000*
GR	5.310	*0.589*	5.760	*0.868*	6.220	*0.001*
GINI	5.500	*0.285*	7.770	*0.000*	7.870	*0.000*
JM	8.810	*0.000*	6.910	*0.011*	6.450	*0.000*
MDL	6.420	*0.002*	6.710	*0.025*	6.770	*0.000*
MGINI	6.540	*0.001*	9.610	*0.000*	9.140	*0.000*
RLV	**4.340**	*0.076*	6.570	*0.078*	6.140	*0.001*
RLF	12.700	*0.000*	11.400	*0.000*	10.070	*0.000*
SGAIN	8.240	*0.000*	**4.400**	*0.005*	7.220	*0.000*
SGINI	5.680	*0.151*	6.270	*0.218*	6.900	*0.000*
WOE	11.180	*0.000*	5.660	*0.968*	6.120	*0.005*

Table 4.11 *Hyper-heuristic using Attribute Entropy and 5 heuristics compared to Standard Algorithms using Ranking Values of Multiple Data Set Experiments*

	ca, de, ec, wi		co, cr, io, sp		ye, vo, he, fl	
HH-12	5.460	*p-value*	4.560	*p-value*	**3.660**	*p-value*
CHI	4.420	*0.015*	6.860	*0.000*	7.020	*0.000*
IG	**3.860**	*0.000*	7.380	*0.000*	7.120	*0.000*
GR	5.490	*0.944*	5.240	*0.130*	6.850	*0.000*
GINI	5.130	*0.441*	8.600	*0.000*	8.220	*0.000*
JM	8.820	*0.000*	6.700	*0.000*	6.810	*0.000*
MDL	6.580	*0.011*	6.720	*0.000*	6.790	*0.000*
MGINI	7.240	*0.000*	10.280	*0.000*	8.360	*0.000*
RLV	4.640	*0.048*	6.310	*0.000*	7.080	*0.000*
RLF	12.600	*0.000*	11.770	*0.000*	9.850	*0.000*
SGAIN	8.060	*0.000*	**4.210**	*0.410*	6.560	*0.000*
SGINI	5.440	*0.962*	6.350	*0.000*	7.220	*0.000*
WOE	11.250	*0.000*	5.970	*0.002*	5.290	*0.001*

Table 4.12 *Hyper-heuristic using Attribute Entropy and 12 heuristics compared to Standard Algorithms using Ranking Values of Multiple Data Set Experiments*

As in the experiments presented in 4.1, both hyper-heuristics managed to get the best ranking on [**ye, vo, he, fl**]. Furthermore, all the standard decision-tree building methods performed statistically significantly worse than both hyper-heuristics for this group of data sets. HH-5 performed poorly on the other two data set groups as the overall rankings it achieved are statistically significantly worse than those of the top ranked methods in each group. HH-12 also performed very poorly on [**ca, de, ec, wi**] but its overall ranking on [**co, cr, io, sp**] is not significantly different from that of the top ranked method (SGAIN). The average predictive accuracy values for all of these results can be found in tables A.11 – A.12 in the Appendix.

4.4 Hyper-heuristic Rules using Maximum Conditional Entropy

4.4.1. Problem State Representation & Choice of Splitting Method

When faced with a data partition to be split, the hyper-heuristics described so far have inspected each splitting attribute individually while ignoring any information about the class attribute. This last hyper-heuristic differs from the previous ones in that it does use information about the class attribute to make its decision on how to split the data. The rules used by this hyper-heuristic look at the relationship between each splitting attribute and the class attribute. More specifically, we use the maximum conditional entropy (MCE) of the class attribute after splitting the data using the candidate splitting attribute. This is formally described in equation 4.1.

$$\mathrm{MCE}(\boldsymbol{d},\ \boldsymbol{y}) = \max_i\{\ H(\boldsymbol{d},\ \boldsymbol{c} \mid \boldsymbol{y}_i)\ \}$$

where, $H(\boldsymbol{d},\ \boldsymbol{c} \mid \boldsymbol{y}_i)$ is the entropy of class attribute $\boldsymbol{c}$ in data partition $\boldsymbol{d}$ where $\boldsymbol{d}$ is made up of all the instances that have value $\boldsymbol{i}$ for attribute $\boldsymbol{y}$.

Equation 4.1

The MCE for splitting attribute $\boldsymbol{y}$ represents the resultant worst entropy value of the class attribute after splitting the data partition using attribute $\boldsymbol{y}$. We decided to involve the class attribute in the hyper-heuristic's decision-making process in the hope of helping it choose splitting heuristics that will ultimately lead to decision trees of a higher predictive accuracy. A hyper-heuristic that decides on how to split the data when faced with a data partition $\boldsymbol{d}$ is represented as a set of $\boldsymbol{m}$ rules in the following manner:

```
IF          $x_1 < low\_MCE_1$ AND
            $y_1 > high\_MCE_1$ THEN use heuristic $h_1$
ELSE IF     $x_2 < low\_MCE_2$ AND
            $y_2 > high\_MCE_2$ THEN use heuristic $h_2$
...
ELSE IF     $x_{m-1} < low\_MCE_{m-1}$ AND
            $y_{m-1} > high\_MCE_{m-1}$ THEN use heuristic $h_{m-1}$
ELSE use heuristic $h_m$
```

where, $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are both percentage values ranging from 0 to 100,

low_MCE_i and $high\_MCE_i$ are thresholds for maximum conditional entropy,

$\boldsymbol{h}_i$ is the heuristic to use to sort the list of candidate splitting attributes should rule $\boldsymbol{i}$ be triggered.

This hyper-heuristic rule set works in a similar manner to the rule set described in 4.3.1 with the difference that we use the MCE of the class attribute brought about by each splitting attribute instead of the entropy of each splitting attribute. This means that each conditional represents a problem state defined by three bins: attributes with low MCE, medium MCE and high MCE. When a data partition needs to be split, the MCE of each splitting attribute is calculated afterwhich the conditional of each rule in the hyper-heuristic is compared to the current problem state on the basis of the MCE values calculated. If one of the rule conditionals matches the problem state, the heuristic dictated by that rule is used to split the partition. If no match is found, the default heuristic $\boldsymbol{h_m}$ is used. One drawback of this method for representing the problem state is that the size of the partition with the maximum conditional entropy is not taken into consideration.

4.4.2 Genetic Algorithm

The genetic algorithm used to search for good hyper-heuristics that use MCE values remains almost entirely unchanged from the one that searches for hyper-heuristics that use attribute entropy values. The encoding is similar to the one used in 4.3.2:

$$\boldsymbol{l_1, x_1, u_1, y_1, h_1, \ l_2, x_2, u_2, y_2, h_2, \ldots, l_{m-1}, x_{m-1}, u_{m-1}, y_{m-1}, \ h_{m-1}, \ h_m}$$

where for any rule i:

- $\boldsymbol{l_i}$ and $\boldsymbol{u_i}$ are real values defining three ranges for low, medium and high MCE,
- $\boldsymbol{x_i}$ and $\boldsymbol{y_i}$ are integer values representing the percentage of attributes that have low MCE and high MCE respectively,
- $\boldsymbol{h_i}$ is an integer value indexing the heuristic to be used to split the data should rule $\boldsymbol{i}$ be triggered.

The only difference in this encoding is in the the genes that the define the ranges for MCE values. Since we are only dealing with two-class problems, the MCE value of any two-valued class attribute for any partition can never be greater than 1. For this reason, the genes $\boldsymbol{u_i}$ that define high MCE values can take one of these eleven possible values: {0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.99}. Otherwise, the granularity of the percentage genes and the genes that define the low MCE values is the same as described in 4.3.2.

As before, we ran single and multiple data set experiments where each single data set experiment was run 20 times and each multiple data set experiments was run 100 times, each time varying the hyper-heuristic training set and test set as well as the random seed of the initial population of the genetic algorithm.

4.4.3 Results

4.4.3.1 Single Data Set Experiments

Tables 4.13 and 4.14 show results for HH-5 and HH-12 trained and tested on single data sets. As in 4.2 and 4.3, HH-12 ranked first on the **hearts** data set though the average ranking value is not statistically different from the second best result. HH-5 topped the results table for the **spect** data set. This hyper-heuristic achieved an overall ranking not significantly worse than the best ranked method in 8 of the 12 data sets. HH-12 achieved this on only 3 of the 12 data sets. The average predictive accuracy values for all of these results can be found in tables A.13 and A.14 in the Appendix.

	car		contrac		credit		derma		ecoli		flags	
HH-5	2.350	*p-value*	5.700	*p-value*	4.100	*p-value*	4.350	*p-value*	4.350	*p-value*	6.650	
CHI	3.000	*0.395*	7.050	*0.302*	4.450	*0.699*	5.600	*0.199*	4.800	*0.724*	5.100	*0.156*
IG	**1.750**	*0.375*	7.500	*0.147*	8.250	*0.000*	4.700	*0.742*	2.750	*0.110*	4.400	*0.045*
GR	3.600	*0.150*	5.200	*0.704*	5.400	*0.157*	3.650	*0.488*	4.800	*0.715*	4.650	*0.071*
GINI	**1.750**	*0.375*	7.550	*0.147*	8.350	*0.001*	4.650	*0.764*	3.100	*0.221*	4.600	*0.046*
JM	10.900	*0.000*	6.450	*0.551*	4.400	*0.727*	5.850	*0.171*	3.800	*0.648*	6.550	*0.930*
MDL	3.050	*0.372*	8.050	*0.082*	4.250	*0.871*	6.900	*0.026*	6.000	*0.164*	4.100	*0.014*
MGINI	2.550	*0.782*	8.250	*0.057*	9.650	*0.000*	7.900	*0.005*	4.250	*0.931*	6.000	*0.548*
RLV	2.200	*0.858*	6.100	*0.745*	7.350	*0.001*	4.050	*0.767*	3.100	*0.221*	4.150	*0.013*
RLF	12.600	*0.000*	6.700	*0.457*	11.700	*0.000*	9.200	*0.000*	9.350	*0.000*	10.550	*0.001*
SGAIN	10.050	*0.000*	4.950	*0.540*	**3.100**	*0.237*	**2.950**	*0.119*	**3.000**	*0.230*	4.250	*0.015*
SGINI	3.150	*0.312*	**4.600**	*0.346*	7.200	*0.007*	5.550	*0.243*	5.100	*0.504*	**3.450**	*0.002*
WOE	12.200	*0.000*	6.350	*0.626*	5.600	*0.119*	6.650	*0.039*	7.450	*0.022*	8.650	*0.120*

	heart		ionosphere		spect		votes		wine		yeast	
HH-5	3.500	*p-value*	5.400	*p-value*	**3.150**	*p-value*	3.950	*p-value*	6.100	*p-value*	5.350	*p-value*
CHI	5.850	*0.013*	**3.350**	*0.047*	6.000	*0.007*	4.450	*0.664*	3.950	*0.070*	6.450	*0.356*
IG	7.200	*0.000*	5.750	*0.733*	5.250	*0.050*	4.900	*0.373*	4.350	*0.148*	5.400	*0.968*
GR	6.350	*0.004*	4.100	*0.153*	5.400	*0.068*	**2.850**	*0.235*	**3.050**	*0.009*	7.050	*0.164*
GINI	7.350	*0.000*	4.550	*0.380*	6.150	*0.004*	4.200	*0.812*	3.150	*0.007*	5.700	*0.773*
JM	5.400	*0.064*	6.600	*0.311*	5.600	*0.018*	4.850	*0.419*	4.350	*0.198*	7.100	*0.162*
MDL	6.900	*0.001*	5.100	*0.786*	5.800	*0.028*	4.600	*0.564*	3.550	*0.033*	6.850	*0.255*
MGINI	7.900	*0.000*	8.450	*0.013*	7.400	*0.001*	3.350	*0.553*	5.000	*0.376*	5.950	*0.633*
RLV	5.700	*0.034*	4.000	*0.148*	6.550	*0.006*	4.650	*0.540*	4.350	*0.151*	6.650	*0.331*
RLF	4.000	*0.617*	9.400	*0.001*	4.250	*0.320*	11.500	*0.000*	10.150	*0.001*	8.650	*0.016*
SGAIN	7.500	*0.000*	5.700	*0.786*	4.100	*0.395*	2.900	*0.259*	5.700	*0.755*	**4.550**	*0.506*
SGINI	7.800	*0.000*	5.900	*0.564*	5.550	*0.019*	3.750	*0.858*	3.700	*0.039*	5.400	*0.963*
WOE	**2.150**	*0.093*	6.400	*0.404*	4.250	*0.295*	6.300	*0.063*	7.700	*0.273*	8.500	*0.020*

Table 4.13 Hyper-heuristic using Maximum Conditional Entropy and 5 heuristics compared to Standard Algorithms using Ranking Values of Single Data Set Experiments

	car		contrac		credit		derma		ecoli		flags	
HH-12	2.200	*p-value*	5.850	*p-value*	4.750	*p-value*	3.850	*p-value*	4.550	*p-value*	5.800	*p-value*
CHI	3.700	*0.039*	7.950	*0.080*	5.150	*0.702*	5.350	*0.164*	**2.500**	*0.049*	4.350	*0.132*
IG	**1.800**	*0.484*	7.300	*0.248*	6.500	*0.086*	4.200	*0.740*	3.500	*0.277*	4.950	*0.425*
GR	3.450	*0.079*	5.100	*0.549*	6.350	*0.134*	6.900	*0.015*	4.000	*0.595*	4.500	*0.219*
GINI	2.000	*0.734*	6.050	*0.859*	7.450	*0.013*	4.750	*0.379*	3.700	*0.374*	6.600	*0.453*
JM	10.850	*0.000*	6.400	*0.651*	6.800	*0.089*	5.950	*0.041*	7.450	*0.031*	4.550	*0.232*
MDL	3.100	*0.180*	6.450	*0.645*	6.800	*0.048*	7.250	*0.006*	6.200	*0.228*	**3.200**	*0.014*
MGINI	4.000	*0.043*	7.900	*0.109*	8.700	*0.000*	7.150	*0.006*	3.250	*0.182*	6.900	*0.335*
RLV	2.450	*0.763*	8.500	*0.029*	6.250	*0.167*	4.250	*0.698*	3.700	*0.394*	5.050	*0.472*
RLF	12.800	*0.000*	8.750	*0.043*	11.200	*0.000*	9.000	*0.000*	10.050	*0.000*	7.300	*0.241*
SGAIN	10.150	*0.000*	**3.400**	*0.024*	4.850	*0.927*	**1.950**	*0.031*	3.900	*0.531*	4.150	*0.097*
SGINI	3.100	*0.170*	5.650	*0.862*	6.050	*0.210*	5.350	*0.130*	3.200	*0.154*	4.450	*0.192*
WOE	12.050	*0.000*	5.600	*0.844*	**2.500**	*0.012*	8.950	*0.000*	6.900	*0.084*	7.150	*0.273*

	heart		ionosphere		spect		votes		wine		yeast	
HH-12	**2.150**	*p-value*	7.300	*p-value*	3.550	*p-value*	4.750	*p-value*	5.150	*p-value*	7.500	*p-value*
CHI	7.250	*0.000*	5.700	*0.188*	5.300	*0.063*	**1.900**	*0.000*	4.200	*0.446*	5.600	*0.074*
IG	6.800	*0.000*	5.100	*0.061*	5.300	*0.086*	6.000	*0.239*	4.400	*0.531*	6.100	*0.189*
GR	7.250	*0.000*	**4.000**	*0.002*	5.650	*0.090*	3.300	*0.111*	**2.750**	*0.030*	6.000	*0.209*
GINI	7.650	*0.000*	5.750	*0.152*	5.500	*0.041*	5.800	*0.300*	5.050	*0.937*	5.550	*0.083*
JM	4.300	*0.003*	5.850	*0.193*	4.300	*0.474*	4.800	*0.964*	5.500	*0.794*	7.150	*0.777*
MDL	6.000	*0.000*	4.350	*0.012*	5.650	*0.040*	3.000	*0.070*	3.750	*0.247*	6.500	*0.413*
MGINI	8.400	*0.000*	7.700	*0.728*	7.900	*0.001*	4.450	*0.757*	4.750	*0.748*	6.550	*0.439*
RLV	5.100	*0.000*	5.250	*0.072*	4.500	*0.324*	4.150	*0.521*	4.400	*0.517*	6.450	*0.393*
RLF	3.750	*0.025*	9.900	*0.029*	5.450	*0.144*	11.450	*0.000*	7.500	*0.103*	7.650	*0.910*
SGAIN	7.800	*0.000*	4.700	*0.013*	**2.550**	*0.190*	3.450	*0.174*	4.300	*0.505*	**4.950**	*0.040*
SGINI	8.200	*0.000*	5.700	*0.135*	6.000	*0.009*	4.600	*0.867*	3.000	*0.050*	**4.950**	*0.023*
WOE	2.500	*0.565*	5.500	*0.137*	5.500	*0.090*	6.100	*0.268*	8.000	*0.041*	7.650	*0.908*

Table 4.14 Hyper-heuristic using Maximum Conditional Entropy and 12 heuristics compared to Standard Algorithms using Ranking Values of Single Data Set Experiments

4.4.3.2 Multiple Data Set Experiments

Tables 4.15 and 4.16 show results for HH-5 and HH-12 trained and tested on multiple data sets.

	ca, de, ec, wi		co, cr, io, sp		ye, vo, he, fl	
HH-5	5.250	*p-value*	5.800	*p-value*	**4.850**	*p-value*
CHI	4.450	*0.325*	**5.600**	*0.839*	6.600	*0.076*
IG	4.300	*0.265*	6.600	*0.471*	8.150	*0.003*
GR	5.450	*0.826*	6.350	*0.607*	6.350	*0.187*
GINI	5.900	*0.408*	7.000	*0.237*	8.200	*0.004*
JM	8.750	*0.001*	5.700	*0.928*	5.800	*0.427*
MDL	6.750	*0.138*	6.650	*0.444*	5.750	*0.469*
MGINI	7.200	*0.040*	9.650	*0.000*	9.350	*0.000*
RLV	**4.250**	*0.263*	7.250	*0.244*	6.950	*0.070*
RLF	12.250	*0.000*	11.550	*0.000*	10.500	*0.000*
SGAIN	7.800	*0.018*	5.750	*0.966*	6.700	*0.097*
SGINI	6.150	*0.284*	6.850	*0.323*	5.250	*0.721*
WOE	11.250	*0.000*	6.200	*0.742*	6.450	*0.192*

Table 4.15 Hyper-heuristic using Maximum Conditional Entropy and 5 heuristics compared to Standard Algorithms using Ranking Values of Multiple Data Set Experiments

	ca, de, ec, wi		co, cr, io, sp		ye, vo, he, fl	
HH-12	5.080	*p-value*	4.850	*p-value*	**4.420**	*p-value*
CHI	4.910	*0.686*	6.830	*0.000*	6.670	*0.000*
IG	**3.760**	*0.001*	7.730	*0.000*	6.950	*0.000*
GR	5.780	*0.104*	5.650	*0.109*	6.880	*0.000*
GINI	4.850	*0.580*	8.090	*0.000*	7.630	*0.000*
JM	9.170	*0.000*	6.440	*0.001*	6.850	*0.000*
MDL	6.910	*0.000*	6.190	*0.003*	7.090	*0.000*
MGINI	6.490	*0.001*	10.050	*0.000*	8.690	*0.000*
RLV	4.840	*0.585*	7.110	*0.000*	7.140	*0.000*
RLF	12.710	*0.000*	11.620	*0.000*	9.880	*0.000*
SGAIN	7.670	*0.000*	**4.560**	*0.516*	7.270	*0.000*
SGINI	5.460	*0.363*	6.310	*0.001*	6.630	*0.000*
WOE	11.100	*0.000*	5.520	*0.156*	4.720	*0.564*

Table 4.16 Hyper-heuristic using Maximum Conditional Entropy and 12 heuristics compared to Standard Algorithms using Ranking Values of Multiple Data Set Experiments

As in almost all of the previous experiments, both HH-5 and HH-12 managed to get the best overall ranking on [**ye, vo, he, fl**], though in both cases the average ranking value is not statistically different from the second best ranking value. As regards to the other two data set groups, HH-5 managed to achieve an overall ranking not significantly different from the best ranked method of each group. HH-12 did not achieve this for data set group [**ca, de, ec, wi**]. The average predictive accuracy values for all of these results can be found in tables A.15 – A.16 in the Appendix.

4.5 Discussion

In this section we discuss the performances of the various hyper-heuristics presented in this chapter. We base our discussion on four different criteria:

a) **Ranked 1st**: the number of data sets (or, in the case of multiple data set experiments, data set groups) on which the hyper-heuristic got the best overall ranking when compared to the standard methods.

b) **Stat Sign Better**: the number of data sets (or data set groups) on which the hyper-heuristic got a result that is statistically significantly better than all the standard algorithms.

c) **NSDTB**: Not Significantly Different To the Best Ranked method – the number of data sets (or data set groups) on which the hyper-heuristic's overall ranking was not significantly different to that of the best ranked method, assuming a confidence level of 90%.

d) **Within Top 3**: the number of data sets (or data set groups) on which the hyper-heuristic manage to rank within the top 3 methods.

We start by looking at how the various hyper-heuristics performed in the single data set experiments. In these experiments we attempted to evolve specialized hyper-heuristics geared towards building accurate decision trees for one particular data set. We did this by training the hyper-heuristic on just one training set – a training set built from the data set we want to specialize for. In doing so, we hoped to evolve hyper-heuristic rules that are fine-tuned for the make-up of the data set from which the training set was built. This experiment was carried out on 12 different problems.

Hyper-heuristic	Ranked 1st	Stat Sign Better	NSDTB	Within Top 3
HH-5 Attr Left	0	0	6	6
HH-12 Attr Left	2	0	5	7
HH-5 Val Count	1	0	8	5
HH-12 Val Count	1	0	9	9
HH-5 Entropy	0	0	10	9
HH-12 Entropy	1	1	8	7
HH-5 MCE	1	0	8	9
HH-12 MCE	1	0	3	6

Table 4.17 Hyper-heuristics Compared on Single Data set Experiments

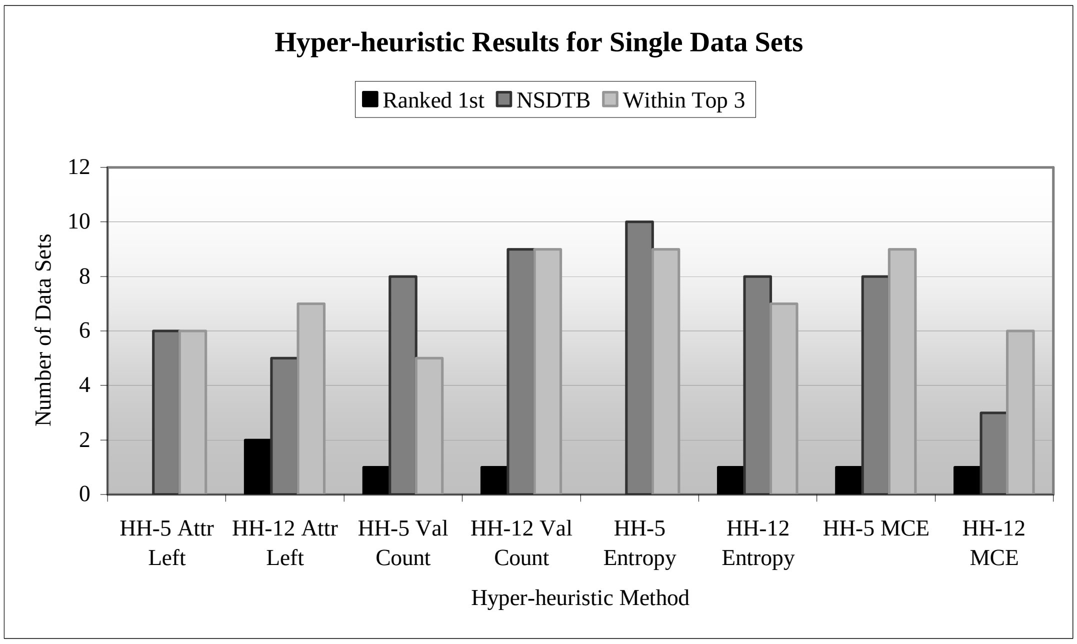

Figure 4.4 Hyper-heuristics Compared on Single Data set Experiments

We evaluated the resultant hyper-heuristic by comparing its performance with those of 12 other standard decision-tree building algorithms on a test set that was never used in the evolutionary training phase of the hyper-heuristic. Each of the 12 standard algorithms uses a different static splitting heuristic. Table 4.17 summarizes the performance of each hyper-heuristic presented in this chapter over all 12 problem cases using the four criteria mentioned above. We also present three of this criteria in the form of a graph in Figure 4.4.

Most of the hyper-heuristics managed to outrank the rest of the standard methods in only 1 of the 12 problem cases. The exceptions are HH-5 Attr Left and HH-5 Entropy which never managed to rank first on any of the problem cases, though the overall performance of HH-5 Entropy was not significantly different to that of the best performing method in 10 of the 12 cases. HH-12 Attr Left ranked first in 2 cases but was significantly worse than the best performing method on 7 other cases.

The hyper-heuristics Val Count, Entropy and MCE all gave very good performances on the **hearts** data set. The performance of HH-12 Entropy is particularly impressive for this data set since it managed to rank first as well as achieve an overall ranking significantly better than the runner-up method (WOE). What we find really interesting though are the results achieved by HH-5 Val Count. The best performing standard method for the **hearts** data set uses the weight of evidence heuristic. HH-5 Val Count does not have this heuristic at its disposal[1] yet it still managed to rank first while achieving a result that is slightly better than that of WOE. In this case, the hyper-heuristic is at a disadvantage since it does not have the splitting heuristic that is best suited for this data set. Yet it still managed to top the results table by using the pool of heuristics made available to it. This suggests that HH-5 Val

[1] Recall that HH-5 can only choose from heuristics information gain, gain ratio, J-measure, relief and symmetric gain.

Count is indeed adapting the splitting heuristic according to the problem state in an effective manner.

These results suggest that problems similar to **hearts** would be well tackled by hyper-heuristic rules that take into consideration the number of distinct values of each attribute - Entropy and MCE are both heavily affected by this property. However there still remains the problem of identifying a template for problem cases of this nature. What is it that makes the **hearts** problem ideal for using such a hyper-heuristic? What is it that sets this data set apart from the rest of the data sets? All in all, these results suggest that in most cases there is no significant improvement to be achieved should one decide to use a hyper-heuristic trained on a single data set instead of one of the standard methods.

Hyper-heuristic	Ranked 1st	Stat Sign Better	NSDTB	Within Top 3
HH-5 Attr Left	1	0	3	3
HH-12 Attr Left	1	0	3	2
HH-5 Val Count	0	0	3	1
HH-12 Val Count	1	1	2	2
HH-5 Entropy	1	1	1	2
HH-12 Entropy	1	1	2	2
HH-5 MCE	1	0	3	1
HH-12 MCE	1	0	2	2

Table 4.18 Hyper-heuristics Compared on Multiple Data set Experiments

We next analyze the results achieved by the hyper-heuristics that were trained and tested on multiple data sets. In this set of experiments we wanted to evolve generalised hyper-heuristics that perform consistently well on a number of problems. We do not expect such a hyper-heuristic to outrank every other standard algorithm on all of the problem cases. But we do expect it to give a better *overall* performance when applied to the same set of problems that were used for training. The goal in this case is to evolve hyper-heuristics that can adapt the heuristics used to build decision trees according to the problem being solved. For this set of experiments we used 3 groups of data sets where each group contains 4 different data sets. The results are presented in Table 4.18 and Figure 4.5.

For all the three groups of data sets, HH-5 Attr Left, HH-12 Attr Left, HH-5 MCE and HH-5 Val Count managed to achieve an overall ranking that is not significantly different to ranking of the best performing method. None of the standard algorithms managed to achieve this. This suggets the robustness and reliability of these hyper-heuristics when trained over multiple data sets. Furthermore, all of the hyper-heuristics except for HH-5 Val Count achieved the best overall ranking on the **[ye, vo, he, fl]** data set group. HH-12 Val Count, HH-5 Entropy and HH-12 Entropy also managed to achieve an overall ranking significantly better than the best performing standard method for this group: WOE. This is especially impressive for HH-5 Entropy since this hyper-heuristic does not have the weight of evidence splitting heuristic at its disposal. In light of all this, we can safely say that there is a clear benefit in using such generalized hyper-heuristics.

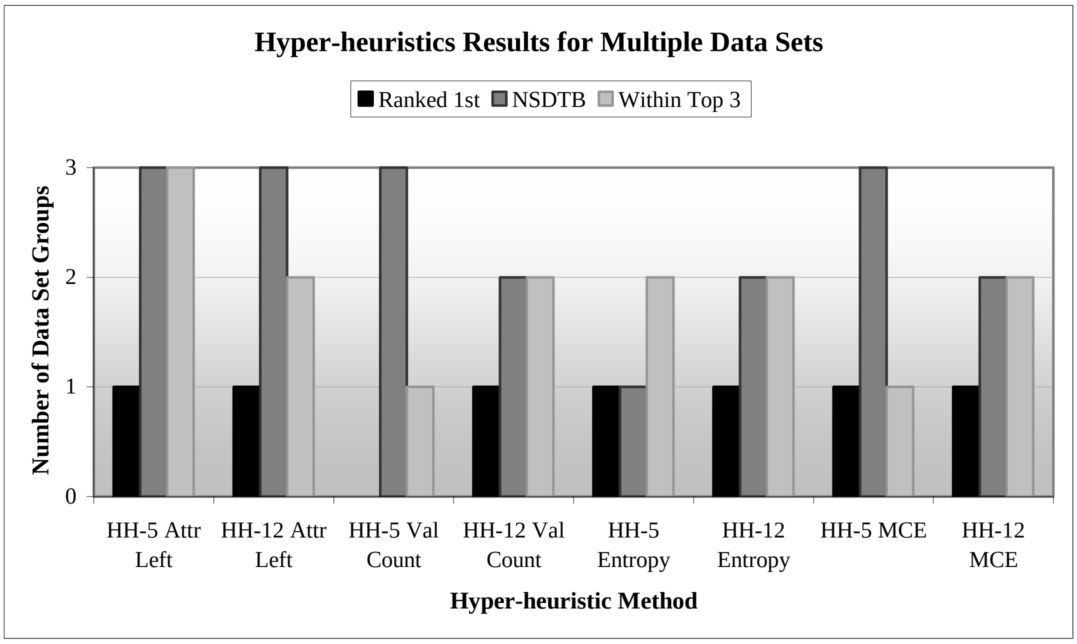

Figure 4.5 Hyper-heuristics Compared on Multiple Data set Experiments

It is also interesting to note that the average predictive accuracy achieved by HH-5 Attr Left on the **flags** data set is better than any of the standard algorithms (see A.3). The same kind of hyper-heuristic did not manage achieve this when it was trained solely on the **flags** data set. As strange as it sounds, training the hyper-heuristic on more than one problem can create rules that perform better on a particular problem than rules which have been specialized for that same problem. This corroborates with the arguments put forward in the previous chapter which state that training the hyper-heuristic on multiple data sets helps evolve rules that are less likely to overfit the make-up of any one particular training set. Such rules will be better at adapting the heuristic to be used according to the problem case that is to be solved.

Chapter 5

Experiments with Synthetic Data Sets

In the previous chapter we presented various hyper-heuristics for decision-tree induction. We compared the performance of these hyper-heuristics to standard decision tree building algorithms on a number of real data sets. The results of these experiments indicate that training with a pool of multiple data sets yields a hyper-heuristic that creates decision trees of a higher predictive accuracy than a hyper-heuristic that has been trained on just one data set. It seems that training on multiple data sets helps evolve hyper-heuristics that are better at adapting the heuristic applied according to the problem state. This notion is further explored in this chapter.

We present experimental results that help analyse the relationship between the predictive accuracy of our hyper-heuristics and the data sets it uses for training. For this set of experiments we decided to use synthetic data sets instead of real ones. The main reason for this is that real data sets from different domains tend to vary considerably in terms of size, type of attributes, class distribution and complexity of the underlying classification model. Such variation can introduce unwanted biases in the results of our experiments, making it harder to draw any conclusions.

Another reason for using synthetic data sets is that in the second set of experiments presented in this chapter we investigate the relationship between the variety of class distributions in the pool of training data sets and the resultant hyper-heuristic. Custom-building synthetic data sets allows one to generate a pool of data sets with different underlying classification models that are similar in all properties (such as size, attribute types, complexity, etc.) except for one (in our case, class distribution). This made synthetic data sets an ideal choice for this set of experiments.

5.1 Correlation between Number of Training Data Sets and Performance of Resultant Hyper-heuristic

5.1.1 Experimental Setup

In chapter 4 we tested our hyper-heuristics using two different experimental setups. The first setup uses only one training data set to evolve the hyper-heuristic rules while the second setup uses a set of four training data sets. The results strongly suggest that the latter method yields hyper-heuristic rules that are better at adapting the splitting heuristic to be applied according to the partition that needs to be split. In this section we further investigate this notion by analyzing the relationship between the number of training data sets used and the performance of the resultant hyper-heuristic.

Throughout these experiments we vary the number of training data sets used to evolve each hyper-heuristic while keeping other factors fixed. We evolve hyper-heuristics of the kind described in 4.4. This type of hyper-heuristic comprises of four rules that use maximum conditional entropy to characterize the problem state. For these experiments all 12 splitting heuristics (see 3.1.1) are made available to the hyper-heuristics during the training phase. The genetic algorithm described in 4.4 is used to search for a set of hyper-heuristic rules.

We compare 8 different hyper-heuristics: *HH1*, *HH2*, ..., *HH8* where *HHx* means that $\boldsymbol{x}$ different data sets were used for training hyper-heuristic *HHx*. As in previous experiments, each one of the $\boldsymbol{x}$ data sets is split into a training set and a test set. The training sets are used in the evolutionary search that produces the hyper-heuristic. The test sets are used to compare the performance of this hyper-heuristic to the standard decision-tree building algorithms.

We perform 10 rounds of experiments. In each round we evolve all of the 8 different types of hyper-heuristics afterwhich they are compared to the standard methods on the test sets. We use a set of 8 different synthetic data sets for each of the 10 rounds so that HH1 is trained on one of the data sets, HH2 on two, HH3 on three, etc. In each round we repeat each experiment 10 times. This effectively means that for each hyper-heuristic type we run 100 experiments (10 runs per 10 rounds).

Using a unique set of 8 synthetic data sets for each round means that we had to generate 80 different synthetic data sets in all. The aim of these experiments is to investigate how the performance of the hyper-heuristic changes according to the number of training data sets used to create the hyper-heuristic. We want our results to be as unbiased as possible from other unrelated factors. For this reason, all of the 80 data sets utilized in these experiments adhere to the following criteria:

- each data set contains 300 instances and 20 attributes,

- all the attributes in each data set are discrete with a value count ranging between 2 and 8,
- the class attribute of each data set can only have two values,
- each data set can be perfectly represented by a decision tree of depth 4.

In this way, we create 80 different, relatively simple problem cases that are as similar as possible to each other in terms of complexity and size.

5.1.2 Results & Discussion

As in the previous chapter, we use rankings instead of predictive accuracy to compare the hyper-heuristic methods to the standard methods since ranking values are non-parametric. Table 5.1 shows the average ranking obtained by each type of hyper-heuristic in each round as well as the overall average ranking for all 10 rounds. These results are also presented in the form of a bar graph in Figure 5.1.

	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8
Round 1	2	2	1	3	4	5	5	3
Round 2	4	2	1	1	3	2	1	2
Round 3	7	3	1	3	4	5	1	1
Round 4	3	4	2	4	4	4	2	4
Round 5	2	3	2	6	1	3	5	4
Round 6	3	3	4	4	2	5	5	4
Round 7	9	2	2	3	3	3	2	4
Round 8	4	5	2	5	3	3	2	1
Round 9	1	1	1	1	4	2	1	2
Round 10	2	4	2	2	3	4	2	3
Average	**3.7**	**2.9**	**1.8**	**3.2**	**3.1**	**3.6**	**2.6**	**2.8**

Table 5.1 *Ranking Values of Results for Hyper-heuristics HH1-HH8 on Synthetic Data Sets*

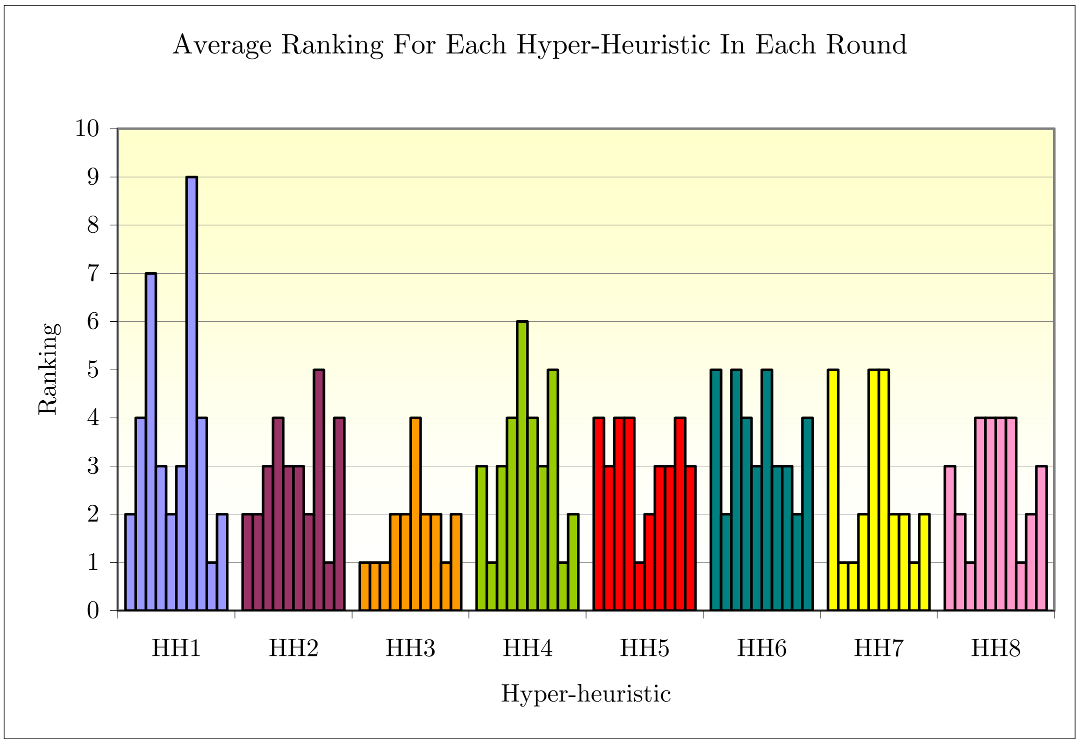

Figure 5.1 *Ranking Results for Hyper-heuristics HH1-HH8 on Synthetic Data Sets*

The graph in figure 5.1 indicates that there is some improvement in performance as more training data sets are used in the learning phase of the hyper-heuristic. The best overall results were achieved when 3 data sets were used for training while the worst overall results were achieved after training with just 1 data set. There is a steady improvement in performance from HH1 to HH2 and from HH2 to HH3. However, when more than 3 training data sets are used, the hyper-heuristics' performance deteriorates. One possible reason for this dip in performance is that a bigger set of hyper-heuristic rules is needed as more and more data sets are used for training and testing. Recall that all the hyper-heuristics in these experiments use a rule set of size 4. This rule set size seems to work best when 3 data sets are used for training. This could mean that the number of different problem states that require a different splitting heuristic in 3 unique data sets can easily be covered by a rule set of this size When more than 3 data sets are used, a bigger rule set is likely to be needed to accommodate the wider variety of problem states that the hyper-heuristic can encounter.

Applying the Pearson Correlation Coefficient to the number of training data sets and the overall average ranking on all eight hyper-heuristics gives us a value of -0.121. The same statistic applied to the first five hyper-heuristics (HH1-HH5) gives a value of **-0.203**[1]. This indicates a small degree of correlation between the two. We suspect that this correlation will become much stronger if the rule set size of the hyper-heuristic is increased in proportion with the number of training data sets used.

5.2 Correlation between Variety of Class Distribution in Training Data Sets and Performance of Resultant Hyper-heuristic

5.2.1 Experimental Setup

The hyper-heuristic's success in adapting the heuristic according to the problem state depends on its ability to distinguish between problem states that require different heuristics. The problem state in our case is the partition that needs to be split. In section 5.1 we provide experimental evidence that indicates some degree of correlation between the number of training data sets used to create a hyper-heuristic for decision-tree building algorithms and its resultant performance on unseen data sets. Supplying more than one training data set to the learning process of the hyper-heuristic helps it discover rules that are better at identifying partitions that require different splitting heuristics. More training data sets means exposure to a wider spectrum of problem states for the hyper-heuristic – this makes for a "richer" learning phase for the hyper-heuristic. This also prevents the hyper-heuristic from overfitting the specific make-up of one particular training data set.

[1] The correlation is negative as a smaller ranking value indicates a better performance.

However there still remains the question of which set of partition features does the hyper-heuristic actually need to look at so that it is able to distinguish between these different problem states. In chapter 4 we tried to answer this question by presenting four hyper-heuristics that each look at the partition that needs to be split through a different lens. In the next set of experiments we try to answer the same question using a different approach.

The hyper-heuristic rules used in 5.1 use maximum conditional entropy to characterize the problem state. An increase in the number of training data sets brought about an increase in the quality of the hyper-heuristic rules. The hyper-heuristic must be latching onto one or more features of the data partition that needs to be split in order for it to be able to distinguish between one problem state and another. One such possible feature could be the distribution of the class values in the data set. Partitions with different class distributions might be better dealt with using different splitting heuristics. Class distribution affects the maximum conditional entropy – so such a feature would be, to some extent, indirectly captured by our hyper-heuristic rules.

In this section we present experiments that investigate the idea that using training data sets with different class distributions provides for a richer and better learning phase for our hyper-heuristics. We do this by running 10 rounds of experiments that each compare 5 different types of hyper-heuristics: HHC1, HHC2, HHC3, HHC4 and HHC5. The hyper-heuristic rules are the same as the ones used in 5.1. The only thing that varies from one type of hyper-heuristic to another is the set of training data sets used in the learning phase. Table 5.2 lists the synthetic data sets used to train the hyper-heuristics while table 5.3 illustrates which data sets are used in the learning phase of each hyper-heuristic.

Synthetic Data Set	Class Distribution	
	Negative Instances	Positive Instances
SDS1	90%	10%
SDS2	70%	30%
SDS3	50%	50%
SDS4	30%	70%
SDS5	10%	90%

Table 5.2 *Synthetic Data Sets Class Distributions*

Hyper-heuristic	Training Data Sets
HHC1	SDS1
HHC2	SDS1, SDS2
HHC3	SDS1, SDS2, SDS3
HHC4	SDS1, SDS2, SDS3, SDS4
HHC5	SDS1, SDS2, SDS3, SDS4, SDS5

Table 5.3 *Hyper-heuristic Type Details*

We recognize at this point that SDS1 and SDS5, as well as SDS2 and SDS4 are equivlanet data sets since the class labels are symmetrical as regards to the decision-tree building algorithm. As in 5.1, each round of experiments uses a different set of synthetic data sets and each hyper-heuristic type is trained and tested 10 times. All of the 50 synthetic data sets utilized in these experiments also conform to the specifications mentioned in 5.1.1. so that the results are not biased by factors unrelated to class distribution.

5.2.2 Results & Discussion

Table 5.4 shows the average ranking obtained by each type of hyper-heuristic in each round as well as the overall average ranking for all 10 rounds. We again present these results in the form of a graph in figure 5.2.

	HHC1	HHC2	HHC3	HHC4	HHC5
Round 1	4	5	2	6	1
Round 2	1	3	3	1	3
Round 3	8	4	3	3	4
Round 4	2	2	1	1	5
Round 5	3	2	4	4	1
Round 6	3	3	4	3	3
Round 7	6	2	3	1	5
Round 8	5	3	2	3	2
Round 9	2	1	1	3	3
Round 10	4	2	2	4	5
Average	**3.8**	**2.7**	**2.5**	**2.9**	**3.2**

Table 5.4 *Ranking Values of Results for Hyper-heuristics HHC1-HHC5 on Synthetic Data Sets*

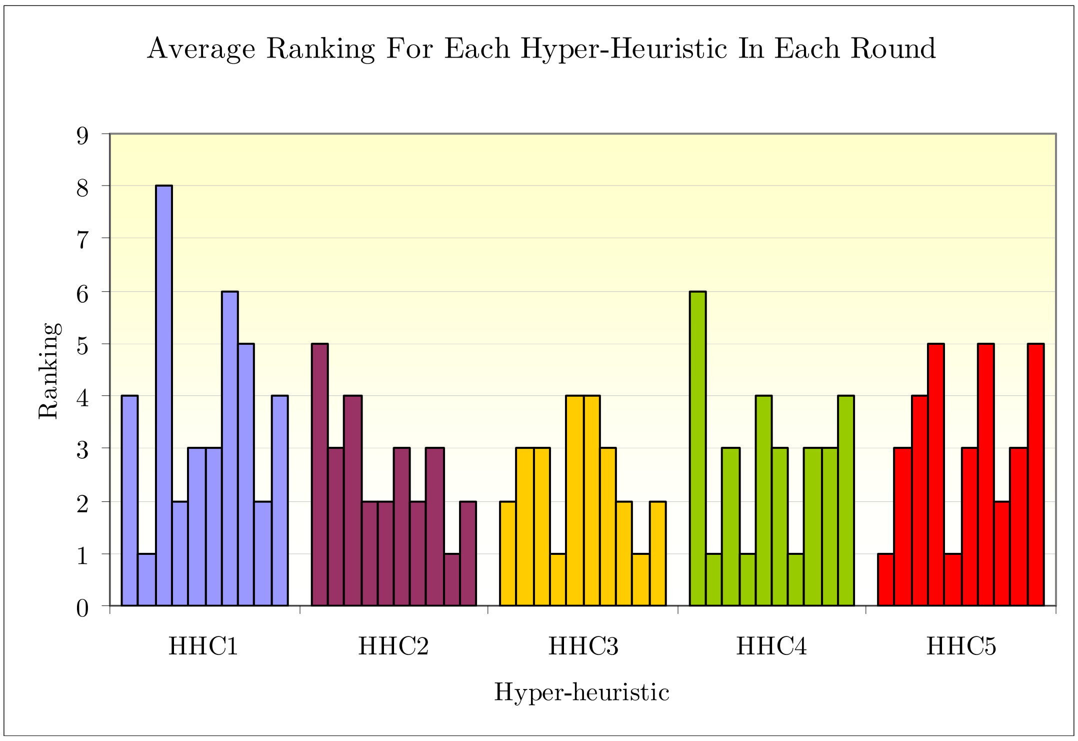

Figure 5.2 *Ranking Values of Results for Hyper-heuristics HHC1-HHC5 on Synthetic Data Sets*

There are many similarities between these results and the previous ones. The best and worst overall results were achieved by the hyper-heuristics that use 3 training data sets and 1 training data set respectively. Again, there is an improvement when the second data set is introduced and when the third data set is introduced. As soon as the fourth data set is introduced, the hyper-heuristic's performance seems to deteriorate. In 5.1.2 we attributed this to the relatively small size of the rule set used by the hyper-heuristic.

This time, when we apply the Pearson Correlation Coefficient to the number of training data sets and the overall average ranking on the five types of hyper-heuristics we get a value of **-0.312**. Recall the the correlation value we got in the previous experiments was -0.203. There is a stronger correlation between the performance of the hyper-heuristic and the number of training data sets when the training data sets have different class distributions. This suggests that using a variety of class distributions in the training phase results in a stronger hyper-heuristic. The hyper-heuristic rules must be exploiting this feature when trying to decide which splitting heuristic to apply when faced with a parititon to be split.

Chapter 6

Conclusion

This thesis has presented what is, to the best of our knowledge, the first attempt at applying the hyper-heuristic paradigm to decision tree building algorithms. We have shown how this is possible through the use of a simple rule set that is embedded within the framework of the typical decision tree building algorithm that creates a tree in a top down fashion. This rule set maps problem states to heuristics and is triggered every time the algorithm needs to create a new node in the tree. The problem state is the data partition left from the training set at that point in the developing tree, while the heuristic is the method by which an attribute is chosen from the ones present in the data partition for the purpose of creating the node in the decision tree. We start by specifying the main contributions of this thesis, after which we list four possible ways in which this work can be extended for the future.

6.1 Contributions

6.1.1 Problem State Representation

The rule set is critical to the performance of the resultant decision tree as it represents the brain of the hyper-heuristic. For this reason, a considerable portion of this thesis was dedicated to the different forms such a rule set can assume. We have presented five different ways in which the problem state could be represented in the hyper-heuristic rule set:

a. partition size in terms of number of instances (3.4 and 3.5),
b. number of attributes left in the partition (4.1),
c. value count of attributes (4.2),
d. entropy of attributes (4.3),
e. maximum conditional entropy of class attribute (4.4).

Experimental evidence shows that hyper-heuristic rules that use some kind of information on the attributes left in the partition (i.e. b, c, d and e) yield more accurate decision trees than rules that simply use the size of the partition (i.e. a) while ignoring information about the attributes. One possible reason for this is because the rule set that uses the size of the partition is much bigger than the four other types of rule sets. This makes it harder for the evolutionary search algorithm to converge to a good set of rules since the search space is so much bigger. However, we believe that the main reason for this disparity in performance is because information about attributes is more relevant than information about the size of the partition to the task of deciding on which heuristic to use to create a tree node from the same partition.

6.1.2 Heuristics for Choosing Splitting Attributes

A hyper-heuristic would typically have a pool of low-level heuristics at its disposal so that it can choose which heuristic to use according to the state of the problem being solved. We have tested three different ways in which our hyper-heuristic can adapt the method for choosing a splitting attribute given a data partition:

a. the hyper-heuristic always uses the same heuristic for sorting attributes while adapting the ranking of the chosen attribute (3.4),
b. given a set of 2 heuristics for sorting attributes, the hyper-heuristic adapts both the heuristic for sorting attributes as well as the ranking of the chosen attribute (3.5),
c. given a set of 5 or 12 heuristics for sorting attributes, the hyper-heuristic adapts the sorting heuristic while always choosing the first ranked attribute (4.1 - 4.4).

Experimental evidence suggests that always choosing the first ranked attribute while adapting the heuristic for sorting attributes yields the most accurate trees from the three strategies just mentioned. We found no difference in performance between stategy a and b. As regards to strategy c, our experiments show no signficant difference in performance between hyper-heuristics that use a pool of 5 heuristics and hyper-heuristics that use a pool of 12 heuristics. We have also highlighted instances of when the hyper-heuristic managed to perform as well as the standard decision tree building algorithm that uses the best available heuristic for the data set being used for testing, even when the hyper-heuristic did not have this heuristic available to use. This proves that the hyper-heuristic is capable of exploiting its available pool of heuristics in an effective way regardless of the quantity and quality of these heuristics.

6.1.3 Specialized and Generalized Hyper-heuristics

We discussed two different applications for each of the hyper-heuristics presented in this thesis: specialized hyper-heuristics and generalized hyper-heuristics. Specialized hyper-heuristics are trained on a single data set from a particular problem domain while generalized hyper-heuristics are trained on multiple data sets from different domains. Specialized hyper-heuristics are meant to be tailored for the particular problem domain they have been trained on and are thus expected to outperform standard non-adaptive methods on data sets coming from this domain. On the other hand, generalized hyper-heuristics are only expected to give a performance that is at least competitive to the best performing method on every problem case they were trained on.

All the results of our experiments indicate that our hyper-heuristics are not suited for specialization - in most occasions the hyper-heuristic's performance on the data set it was specialized for was not significantly better than that of the best performing standard algorithm. We believe that this is due to the hyper-heuristic overfitting the specific make-up of the data set it uses for training. This notion is supported by the more promising results achieved by the generalized hyper-heuristics in Chapter 4. In this case, we managed to achieve results that were very competitive to the best standard methods and in some cases we even achieved a significantly better overall performance than all of the standard methods.

In Chapter 5 we investigated the link between the number of training data sets and the performance of the resultant hyper-heuristic through the use of controlled experiments utilizing synthetic data sets. These experiments confirm that increasing the number of training data sets results in a hyper-heuristic that yields more accurate trees. We also noticed that this correlation weakens as the number of training data sets increases past a certain threshold. We believe that this can be circumvented by increasing the number of rules in the hyper-heuristic rule set so that it is able to accommodate a wider variety of problem states. Further experimentation revealed that this correlation becomes stronger if the data sets used for training contain different class distributions. This implies that the hyper-heuristic could be indirectly using the class distribution of the data partition in order to make a choice on which heuristic to use to split that partition.

6.2 Future work

6.2.1 Searching for Good Hyper-heuristics

Our decision to use genetic algorithms to search for good hyper-heuristics was based on the proven track record of evolutionary algorithms in the field of hyper-heuristics (Smith, 1985; Syswerda, 1991; Fang et al, 1993; Shing and Parker, 1993; Fang et al, 1994; Langdon, 1995; Dorndorf and Pesch, 1995; Fleurent and Ferland, 1996; Reeves, 1996; Schaffer and Eshelman, 1996; Corne and Ogden, 1997; Hart and Ross, 1998; Terashima-Marín et al, 1999; Cowling et al, 2002b; Ross et al, 2003; Vázquez-Rodríguez et al, 2007a) as well as on the results of some preliminary experiments we carried out that compare the performance of our genetic algorithm to that of a simple hillclimbing algorithm. Possible future work could involve using other types of advanced search techniques such as simulated annealing or tabu search.

6.2.2 Analysis of Fitness Landscape

The search space of our genetic algorithm consists of the set of all possible hyper-heuristic rule sets. It might be interesting to analyse the fitness landscape (Jones, 1995) produced by such a genetic algorithm. Looking at the ruggedness of such a landscape would help us understand the underlying complexity of the search problem faced by the genetic algorithm. Such studies might also yield valuable insight into which genetic operators to use so as to optimise the search process of the genetic algorithm.

6.2.3 Modifying the Size of Hyper-Heuristic Rule Set

In Chapter 5 we noted how the performance of our hyper-heuristic deteriorates as the number of training data sets increase past a certain threshold. A possible future direction could be to analyze how increasing the size of the hyper-heuristic rule set would affect this problem. Another option would be to use messy genetic algorithms (Freitas, 2002) so that the evolutionary process is allowed to automatically choose the size of the rule set.

6.2.4 Characterizing the Problem State using the Class Distribution

The results obtained in chapter 5 indicate a correlation between the variety of class distributions in the pool of training data sets and the performance of the resultant hyper-heuristic. Given these promising results, it would be interesting to experiment with a hyper-heuristic that characterizes the problem state using the class distribution directly. The splitting heuristic would then be adapted according to how balanced or unbalanced the class distribution is in the current partition to be split.

6.2.5 Characterizing the Problem State using a Mixture of Criteria

Our hyper-heuristic rules have always used just one particular feature of the data set (ex. size, or attributes left) to characterize the problem state. Another possibility is to use rules that look at several different criteria of the data partition left to work with. For example, it might be useful for a hyper-heuristic to know both the number of attributes left in the partition to split as well as the maximum conditional entropy of those remaining attributes before making a decision on which heuristic to use to split that data partition. Using different criteria might provide a richer description of the problem state to the hyper-heuristic thus helping it make a more informed decision on which method is best suited to deal with that problem state.

6.2.6 Adapting Discretization and Stopping or Pruning Techniques

In our work, all the continuous attributes in the data sets were discretized before we used them for training and testing. We have also deliberately not used any stopping or pruning so as to prevent these processes from introducing any bias into our results. However, we see no reason why one cannot apply the hyper-heuristic paradigm to these processes as well. The heuristic used for the discretization of a continuous attribute could be adapted according to the distribution of unique values of this attribute. The heuristic for stopping the growth of the decision tree could be adapted according to the data left to work with as well to some features of the developing tree. Similarly, pruning would use some features of the fully grown decision tree and relate them to the training set used to grow that tree so as to choose a suitable heuristic for pruning.

Appendix

A.1 Results for Hyper-heuristics that use Number of Attributes Left

The hyper-heuristics in this section represent the problem state using the number of attributes left. Each column represents pairs of values where the first value represents the average accuracy percentage value while the second value is the resultant p-value when comparing the results of the method to the results of the hyper-heuristic on the same data set.

	car		contrac		credit		derma		ecoli		flags	
HH-5	94.545	*p-value*	63.885	*p-value*	80.725	*p-value*	91.757	*p-value*	91.912	*p-value*	83.000	*p-value*
CHI	94.545	*1.000*	62.128	*0.055*	81.377	*0.712*	90.811	*0.491*	92.500	*0.705*	82.750	*0.920*
IG	**94.689**	*0.801*	62.635	*0.189*	80.145	*0.730*	91.892	*0.929*	92.059	*0.910*	81.500	*0.589*
GR	94.581	*0.951*	**64.696**	*0.354*	81.377	*0.704*	89.459	*0.149*	91.471	*0.756*	84.500	*0.524*
GINI	94.653	*0.846*	62.601	*0.171*	80.000	*0.674*	90.811	*0.533*	92.059	*0.910*	82.000	*0.695*
JM	85.441	*0.000*	63.378	*0.600*	80.942	*0.893*	89.189	*0.095*	91.618	*0.850*	82.750	*0.922*
MDL	94.581	0.951	63.953	*0.942*	80.000	*0.666*	89.595	*0.148*	91.471	*0.751*	82.250	*0.787*
MGINI	94.509	*0.950*	61.655	*0.009*	79.928	*0.632*	88.378	*0.024*	92.353	*0.726*	82.500	*0.836*
RLV	94.364	*0.757*	62.905	*0.283*	80.145	*0.749*	92.432	*0.645*	92.353	*0.740*	83.500	*0.846*
RLF	76.409	*0.000*	63.243	*0.516*	70.362	*0.000*	86.081	*0.000*	87.353	*0.004*	73.500	*0.000*
SGAIN	86.344	*0.000*	64.392	*0.593*	81.232	*0.764*	**93.108**	*0.371*	**93.529**	*0.192*	82.250	*0.751*
SGINI	94.545	*1.000*	63.480	*0.640*	**81.812**	*0.537*	90.811	*0.530*	92.941	*0.434*	**85.000**	*0.418*
WOE	80.672	*0.000*	62.264	*0.136*	81.449	*0.645*	87.297	*0.003*	88.971	*0.172*	80.000	*0.190*
	heart		ionosphere		spect		votes		wine		yeast	
HH-5	67.037	*p-value*	86.528	*p-value*	**82.222**	*p-value*	93.409	*p-value*	88.333	*p-value*	71.174	*p-value*
CHI	57.778	*0.001*	86.111	*0.774*	78.519	*0.136*	**94.659**	*0.176*	90.556	*0.371*	71.007	*0.859*
IG	57.407	*0.000*	87.083	*0.729*	79.012	*0.182*	93.636	*0.813*	90.278	*0.398*	**71.544**	*0.710*
GR	59.444	*0.013*	87.917	*0.393*	79.012	*0.144*	94.091	*0.539*	92.778	*0.040*	71.007	*0.871*
GINI	55.370	*0.000*	87.222	*0.668*	78.765	*0.174*	93.636	*0.818*	90.000	*0.451*	71.443	*0.786*
JM	58.704	*0.002*	85.000	*0.294*	79.259	*0.212*	93.523	*0.902*	91.111	*0.170*	70.168	*0.322*
MDL	59.444	*0.014*	**88.611**	*0.183*	79.506	*0.240*	93.409	*1.000*	90.556	*0.364*	71.443	*0.795*
MGINI	55.370	*0.000*	87.083	*0.721*	76.049	*0.016*	93.750	*0.762*	88.333	*1.000*	70.973	*0.846*
RLV	59.444	*0.010*	87.917	*0.377*	79.753	*0.358*	92.500	*0.383*	89.444	*0.628*	70.772	*0.696*
RLF	65.741	*0.637*	81.111	*0.002*	80.988	*0.492*	86.477	*0.000*	85.000	*0.171*	68.960	*0.018*
SGAIN	60.741	*0.017*	88.333	*0.213*	81.481	*0.743*	94.432	*0.311*	88.611	*0.903*	71.208	*0.973*
SGINI	57.222	*0.001*	88.056	*0.347*	79.012	*0.206*	93.864	*0.670*	**92.778**	*0.059*	71.007	*0.866*
WOE	**70.370**	*0.268*	85.139	*0.445*	81.481	*0.732*	93.182	*0.796*	86.944	*0.513*	69.329	*0.034*

Table A.1 Hyper-heuristic using Number of Attributes Left – Average Accuracy Results for Single Data Sets using 5 heuristics

Tables A.1 and A.2 show results for hyper-heuristics trained and tested on single data sets. The hyper-heuristics in A.1 have a set of 5 heuristics at their disposal while the ones in A.2 utilize a set of 12 heuristics.

	car		contrac		credit		derma		ecoli		flags	
HH-12	94.277	*p-value*	63.116	*p-value*	81.449	*p-value*	92.838	*p-value*	91.912	*p-value*	87.250	*p-value*
CHI	94.538	*0.638*	61.288	*0.096*	82.464	*0.462*	90.811	*0.170*	91.176	*0.546*	88.500	*0.631*
IG	**94.855**	*0.293*	62.758	*0.728*	80.797	*0.658*	91.892	*0.538*	93.529	*0.110*	86.750	*0.843*
GR	94.566	*0.607*	64.706	*0.117*	**83.043**	*0.217*	91.892	*0.483*	91.324	*0.608*	89.000	*0.407*
GINI	94.740	*0.390*	62.838	*0.797*	79.420	*0.218*	89.865	*0.048*	93.088	*0.238*	86.750	*0.834*
JM	84.884	*0.000*	63.434	*0.793*	81.304	*0.921*	90.270	*0.085*	92.941	*0.423*	86.000	*0.585*
MDL	94.566	0.613	62.679	*0.682*	81.014	*0.744*	89.730	*0.039*	90.441	*0.199*	89.500	*0.268*
MGINI	94.364	*0.871*	61.804	*0.232*	78.696	*0.074*	89.189	*0.015*	91.324	*0.554*	85.000	*0.423*
RLV	94.653	*0.497*	63.076	*0.971*	82.246	*0.532*	92.027	*0.580*	92.647	*0.450*	87.500	*0.910*
RLF	75.751	*0.000*	61.765	*0.187*	70.000	*0.000*	87.838	*0.006*	87.206	*0.003*	74.500	*0.000*
SGAIN	86.590	*0.000*	**64.785**	*0.109*	82.681	*0.345*	**93.378**	*0.683*	**94.118**	*0.033*	87.250	*1.000*
SGINI	94.451	*0.755*	63.514	*0.718*	82.971	*0.266*	90.811	*0.217*	91.618	*0.734*	**89.750**	*0.276*
WOE	79.480	*0.000*	61.963	*0.324*	81.304	*0.905*	89.459	*0.016*	89.559	*0.093*	83.750	*0.142*
	heart		ionosphere		spect		votes		wine		yeast	
HH-12	74.074	*p-value*	85.833	*p-value*	78.395	*p-value*	94.773	*p-value*	88.056	*p-value*	69.687	*p-value*
CHI	60.370	*0.000*	84.861	*0.666*	77.572	*0.733*	94.773	*1.000*	86.944	*0.655*	70.097	*0.701*
IG	58.519	*0.000*	86.528	*0.748*	77.572	*0.713*	94.545	*0.804*	86.111	*0.470*	70.805	*0.255*
GR	60.000	*0.000*	87.639	*0.368*	76.955	*0.551*	94.773	*1.000*	87.500	*0.837*	69.016	*0.504*
GINI	57.963	*0.000*	86.389	*0.793*	77.778	*0.799*	94.773	*1.000*	86.944	*0.678*	**70.992**	*0.193*
JM	61.481	*0.000*	84.306	*0.440*	77.984	*0.876*	94.318	*0.647*	**89.167**	*0.679*	70.470	*0.437*
MDL	62.407	*0.000*	**87.778**	*0.330*	78.601	*0.933*	94.773	*1.000*	88.333	*0.911*	69.985	*0.758*
MGINI	57.037	*0.000*	85.833	*1.000*	73.045	*0.035*	95.000	*0.774*	87.500	*0.803*	70.134	*0.666*
RLV	61.852	*0.000*	87.083	*0.563*	76.132	*0.443*	95.000	*0.774*	88.611	*0.819*	70.358	*0.488*
RLF	65.741	*0.017*	80.278	*0.013*	76.543	*0.390*	87.614	*0.000*	79.722	*0.002*	69.239	*0.626*
SGAIN	58.889	*0.000*	87.222	*0.473*	76.749	*0.497*	**95.455**	*0.405*	88.611	*0.802*	70.619	*0.386*
SGINI	57.963	*0.000*	86.944	*0.571*	77.572	*0.733*	95.227	*0.577*	86.389	*0.536*	70.358	*0.499*
WOE	**75.556**	*0.633*	86.806	*0.645*	**79.012**	*0.816*	93.977	*0.325*	81.944	*0.047*	69.314	*0.695*

Table A.2 Hyper-heuristic using Number of Attributes Left – Average Accuracy Results for Single Data Sets using 12 heuristics

Tables A.3 and A.4 show results for hyper-heuristics trained and tested on multiple data sets. The hyper-heuristics in A.3 have a set of 5 heuristics at their disposal while the ones in A.4 utilize a set of 12 heuristics.

	ca, de, ec, wi				co, cr, io, sp				ye, vo, he, fl			
	car	derma	ecoli	wine	contrac	credit	ionosphere	spect	yeast	votes	heart	flags
HH-5	94.017	94.054	91.471	90.556	62.997	80.477	84.804	79.085	68.188	92.727	63.333	85.000
CHI	94.104	89.730	91.029	90.556	62.878	79.284	88.072	78.214	67.651	94.091	58.889	81.000
IG	94.133	91.486	92.353	92.222	62.639	78.517	86.111	77.560	67.919	92.955	58.148	84.000
GR	94.046	90.270	90.882	91.667	64.030	79.710	84.804	76.906	67.181	92.727	62.593	83.000
GINI	94.191	90.270	92.206	88.889	62.520	76.812	87.418	77.560	68.054	92.727	54.444	81.000
JM	84.624	90.000	89.706	89.444	64.030	78.772	84.150	79.085	67.718	92.273	58.889	84.000
MDL	94.017	89.189	90.294	90.556	63.593	79.540	87.092	77.778	67.383	91.591	60.000	81.500
MGINI	93.873	89.189	91.176	89.444	62.520	75.959	84.150	72.331	67.450	92.273	57.037	79.500
RLV	94.075	91.757	92.206	89.722	62.679	78.687	87.092	76.253	68.121	91.818	62.222	84.500
RLF	77.168	86.216	85.294	81.944	62.122	70.418	83.007	79.303	69.329	85.682	64.074	71.500
SGAIN	86.272	94.595	92.941	88.333	63.394	82.268	85.784	79.085	69.664	94.318	61.111	84.000
SGINI	94.046	90.946	91.029	91.111	63.235	77.579	84.804	76.906	68.389	92.955	57.778	83.000
WOE	80.405	89.595	88.088	84.722	63.235	81.586	83.497	77.996	67.987	92.727	68.519	76.000

Table A.3 Hyper-heuristic using Number of Attributes Left – Average Accuracy Results for Multiple Data Sets using 5 heuristics

	ca, de, ec, wi				co, cr, io, sp			
	car	derma	ecoli	wine	contrac	credit	ionosphere	spect
HH-12	94.422	91.892	91.912	90.278	62.323	79.641	87.037	81.834
CHI	94.364	90.135	93.529	91.111	60.682	81.297	87.566	80.247
IG	94.711	92.162	92.206	91.667	60.907	78.951	86.905	79.894
GR	94.480	90.135	90.000	90.556	62.773	79.020	88.095	78.660
GINI	94.595	89.189	92.353	90.833	60.843	77.778	86.243	79.541
JM	85.520	90.811	89.706	90.833	61.068	79.434	86.640	80.071
MDL	94.509	88.919	91.618	91.667	61.873	79.434	88.095	79.541
MGINI	94.162	89.595	90.882	90.278	60.521	76.605	84.524	77.249
RLV	94.538	92.568	92.206	90.000	60.167	78.744	87.302	79.541
RLF	77.168	88.378	86.765	81.667	60.521	69.220	81.349	80.247
SGAIN	86.185	94.054	93.235	89.444	62.838	79.434	87.963	81.305
SGINI	94.393	90.676	91.324	91.389	61.454	80.262	87.037	79.718
WOE	80.838	89.865	88.971	85.278	61.615	81.988	84.656	80.423

	ye, vo, he, fl			
	yeast	votes	heart	flags
HH-12	68.098	95.000	68.889	83.333
CHI	68.635	95.152	58.025	85.667
IG	69.396	95.000	56.543	84.333
GR	68.725	96.061	57.037	86.333
GINI	69.038	95.303	53.580	83.333
JM	69.664	94.394	60.494	82.000
MDL	68.322	95.455	58.272	86.000
MGINI	69.262	95.000	54.074	80.333
RLV	69.128	94.697	57.778	84.333
RLF	68.993	88.939	63.457	75.000
SGAIN	69.485	95.455	57.037	84.000
SGINI	68.859	95.909	55.802	84.333
WOE	68.501	93.636	70.617	82.333

Table A.4 Hyper-heuristic using Number of Attributes Left – Average Accuracy Results for Multiple Data Sets using 12 heuristics

A.2 Results for Hyper-heuristics that use Attribute Value Count

The hyper-heuristics in this section represent the problem state using the value count of the attributes left. Each column represents pairs of values where the first value represents the average accuracy percentage value while the second value is the resultant p-value when comparing the results of the method to the results of the hyper-heuristic on the same data set.

	car		contrac		credit		derma		ecoli		flags	
HH-5	95.145	*p-value*	64.088	*p-value*	**81.087**	*p-value*	93.108	*p-value*	90.882	*p-value*	81.500	*p-value*
CHI	95.202	*0.875*	63.007	*0.376*	79.565	*0.368*	91.622	*0.204*	92.941	*0.127*	82.500	*0.714*
IG	**95.347**	*0.573*	63.345	*0.511*	77.174	*0.024*	92.027	*0.377*	93.235	*0.093*	83.000	*0.552*
GR	95.173	*0.933*	64.189	*0.928*	81.014	*0.963*	92.162	*0.402*	89.853	*0.497*	**84.500**	*0.249*
GINI	95.289	*0.700*	63.514	*0.620*	76.594	*0.008*	91.757	*0.264*	93.235	*0.086*	83.750	*0.333*
JM	84.798	*0.000*	**64.493**	*0.759*	78.841	*0.182*	90.946	*0.112*	90.000	*0.615*	79.750	*0.459*
MDL	95.202	0.873	63.851	*0.843*	78.913	*0.212*	90.811	*0.079*	90.735	*0.916*	81.250	*0.929*
MGINI	95.058	*0.809*	61.453	*0.019*	76.232	*0.004*	90.811	*0.110*	92.647	*0.184*	79.500	*0.458*
RLV	94.971	*0.666*	63.209	*0.449*	78.768	*0.178*	92.973	*0.906*	**93.382**	*0.071*	83.750	*0.356*
RLF	76.445	*0.000*	61.723	*0.029*	69.348	*0.000*	86.486	*0.000*	86.324	*0.012*	76.750	*0.070*
SGAIN	86.590	*0.000*	64.257	*0.896*	80.725	*0.818*	**93.243**	*0.899*	91.765	*0.520*	83.500	*0.379*
SGINI	95.173	*0.936*	64.155	*0.957*	80.145	*0.578*	92.162	*0.429*	91.765	*0.504*	82.500	*0.682*
WOE	78.584	*0.000*	63.277	*0.543*	78.913	*0.208*	89.595	*0.018*	88.971	*0.273*	77.500	*0.129*
	heart		ionosphere		spect		votes		wine		yeast	
HH-5	67.778	*p-value*	87.778	*p-value*	79.630	*p-value*	92.955	*p-value*	87.222	*p-value*	69.396	*p-value*
CHI	56.667	*0.000*	86.528	*0.278*	75.926	*0.301*	93.977	*0.476*	88.056	*0.684*	69.497	*0.913*
IG	60.741	*0.001*	86.944	*0.492*	77.778	*0.638*	93.636	*0.598*	87.778	*0.752*	69.161	*0.771*
GR	56.852	*0.000*	88.611	*0.464*	77.037	*0.435*	93.864	*0.508*	88.611	*0.473*	69.161	*0.784*
GINI	56.852	*0.000*	88.194	*0.763*	77.037	*0.435*	93.636	*0.592*	86.111	*0.557*	69.530	*0.881*
JM	61.852	*0.009*	84.444	*0.019*	80.000	*0.916*	93.523	*0.651*	**91.111**	*0.074*	68.993	*0.681*
MDL	58.519	*0.000*	88.472	*0.544*	76.296	*0.367*	94.205	*0.319*	90.000	*0.182*	68.826	*0.515*
MGINI	55.741	*0.000*	87.222	*0.712*	75.185	*0.212*	93.977	*0.413*	85.833	*0.567*	69.228	*0.857*
RLV	59.259	*0.000*	87.917	*0.925*	78.519	*0.757*	94.091	*0.364*	87.778	*0.745*	69.161	*0.817*
RLF	63.889	*0.159*	80.694	*0.000*	77.778	*0.565*	86.591	*0.001*	79.167	*0.012*	69.027	*0.698*
SGAIN	59.074	*0.000*	**89.028**	*0.239*	79.259	*0.916*	94.205	*0.311*	86.667	*0.768*	69.631	*0.791*
SGINI	57.037	*0.000*	87.917	*0.915*	76.667	*0.387*	**94.659**	*0.154*	86.389	*0.685*	**69.732**	*0.721*
WOE	**67.963**	*0.945*	87.500	*0.845*	**80.000**	*0.909*	93.409	*0.709*	85.833	*0.530*	69.060	*0.739*

Table A.5 Hyper-heuristic using Attribute Value Count – Average Accuracy Results for Single Data Sets using 5 heuristics

Tables A.5 and A.6 show results for hyper-heuristics trained and tested on single data sets. The hyper-heuristics in A.5 have a set of 5 heuristics at their disposal while the ones in A.6 utilize a set of 12 heuristics.

	car		contrac		credit		derma		ecoli		flags	
HH-12	94.364	*p-value*	62.878	*p-value*	79.565	*p-value*	90.270	*p-value*	90.147	*p-value*	83.000	*p-value*
CHI	94.220	*0.680*	62.281	*0.522*	79.565	*1.000*	90.135	*0.926*	**91.912**	*0.277*	82.250	*0.786*
IG	**94.451**	*0.798*	61.566	*0.202*	78.913	*0.618*	91.081	*0.548*	91.471	*0.400*	83.250	*0.931*
GR	94.306	*0.870*	62.878	*1.000*	80.000	*0.734*	90.135	*0.932*	90.588	*0.771*	**85.500**	*0.296*
GINI	94.364	*1.000*	61.963	*0.327*	78.333	*0.322*	89.054	*0.381*	91.029	*0.586*	81.500	*0.515*
JM	84.335	*0.000*	62.560	*0.759*	81.957	*0.079*	89.459	*0.571*	91.471	*0.429*	81.000	*0.445*
MDL	94.364	1.000	62.639	*0.812*	79.130	*0.737*	89.324	*0.530*	91.029	*0.557*	82.750	*0.922*
MGINI	94.162	*0.559*	61.328	*0.188*	77.826	*0.157*	87.432	*0.060*	91.324	*0.435*	79.000	*0.120*
RLV	94.277	*0.807*	62.162	*0.420*	80.362	*0.584*	91.216	*0.505*	91.618	*0.365*	83.750	*0.733*
RLF	77.977	*0.000*	62.043	*0.391*	69.203	*0.000*	85.811	*0.010*	86.912	*0.070*	73.500	*0.000*
SGAIN	85.983	*0.000*	**63.831**	*0.253*	**82.391**	*0.023*	**91.486**	*0.459*	91.765	*0.341*	83.250	*0.902*
SGINI	94.220	*0.680*	62.917	*0.963*	79.493	*0.956*	90.405	*0.926*	90.588	*0.769*	**85.500**	*0.272*
WOE	80.491	*0.000*	62.560	*0.757*	81.377	*0.126*	87.973	*0.141*	90.147	*1.000*	77.500	*0.035*

	heart		ionosphere		spect		votes		wine		yeast	
HH-12	71.667	*p-value*	88.056	*p-value*	79.125	*p-value*	93.636	*p-value*	90.000	*p-value*	**70.705**	*p-value*
CHI	60.556	*0.000*	86.806	*0.501*	76.768	*0.468*	**94.205**	*0.645*	88.333	*0.480*	69.799	*0.253*
IG	61.481	*0.000*	87.639	*0.835*	77.778	*0.635*	93.182	*0.716*	88.889	*0.582*	70.235	*0.565*
GR	60.370	*0.000*	87.500	*0.802*	75.421	*0.188*	93.750	*0.922*	89.722	*0.914*	69.396	*0.163*
GINI	57.407	*0.000*	**89.167**	*0.569*	76.768	*0.484*	92.841	*0.523*	88.611	*0.540*	70.638	*0.935*
JM	61.111	*0.000*	85.833	*0.268*	76.768	*0.468*	93.523	*0.925*	**93.056**	*0.141*	69.497	*0.101*
MDL	60.741	*0.000*	88.056	*1.000*	79.798	*0.797*	93.523	*0.929*	88.333	*0.486*	69.497	*0.187*
MGINI	57.778	*0.000*	85.556	*0.196*	75.421	*0.109*	93.182	*0.711*	86.944	*0.167*	69.899	*0.330*
RLV	61.481	*0.000*	87.917	*0.947*	80.135	*0.714*	93.409	*0.852*	89.444	*0.768*	70.436	*0.753*
RLF	65.556	*0.034*	80.972	*0.002*	**80.808**	*0.509*	87.386	*0.000*	79.444	*0.000*	68.591	*0.013*
SGAIN	58.889	*0.000*	88.611	*0.782*	80.471	*0.583*	93.750	*0.924*	87.778	*0.366*	69.262	*0.096*
SGINI	58.889	*0.000*	87.222	*0.685*	76.431	*0.420*	93.750	*0.928*	89.444	*0.807*	69.866	*0.334*
WOE	**72.037**	*0.897*	87.083	*0.663*	80.135	*0.718*	91.591	*0.160*	89.444	*0.828*	69.765	*0.222*

Table A.6 Hyper-heuristic using Attribute Value Count – Average Accuracy Results for Single Data Sets using 12 heuristics

Tables A.7 and A.8 show results for hyper-heuristics trained and tested on multiple data sets. The hyper-heuristics in A.7 have a set of 5 heuristics at their disposal while the ones in A.8 utilize a set of 12 heuristics.

	ca, de, ec, wi				co, cr, io, sp				ye, vo, he, fl			
	car	derma	ecoli	wine	contrac	credit	ionosphere	spect	yeast	votes	heart	flags
HH-5	94.335	92.770	92.500	86.389	63.268	79.644	86.869	78.788	68.984	93.344	64.550	85.714
CHI	94.263	91.554	94.118	86.806	63.391	78.986	88.131	77.778	70.062	94.075	56.481	84.821
IG	94.465	92.905	93.088	85.972	62.346	78.195	86.616	76.094	70.973	93.831	56.878	83.750
GR	94.306	91.081	92.059	87.500	63.851	78.327	89.015	79.798	69.271	93.912	58.069	86.071
GINI	94.321	91.554	92.941	86.250	62.469	77.470	89.141	77.609	70.542	93.588	55.952	83.750
JM	84.653	91.014	91.544	88.472	63.053	78.788	85.859	78.788	69.799	93.101	60.450	84.107
MDL	94.350	90.676	91.471	86.389	63.667	78.327	87.753	77.273	70.350	93.019	58.333	84.464
MGINI	94.234	90.338	92.794	85.972	61.302	75.428	84.848	74.747	69.871	93.182	53.968	82.679
RLV	94.408	92.703	93.088	85.833	62.684	78.327	89.141	78.451	69.823	93.588	60.450	83.393
RLF	75.621	87.568	88.235	81.667	61.149	68.379	81.313	79.293	69.487	85.471	65.608	74.286
SGAIN	85.795	95.068	93.309	86.528	63.114	80.105	87.879	79.461	70.781	93.669	56.878	81.607
SGINI	94.191	91.284	93.162	86.528	62.991	78.920	87.374	77.441	70.062	93.506	56.481	83.750
WOE	81.358	89.797	89.559	83.056	63.360	79.183	87.121	80.135	69.343	92.532	71.561	81.071

Table A.7 Hyper-heuristic using Attribute Value Count – Average Accuracy Results for Multiple Data Sets using 5 heuristics

	ca, de, ec, wi				co, cr, io, sp				ye, vo, he, fl			
	car	derma	ecoli	wine	contrac	credit	ionosphere	spect	yeast	votes	heart	flags
HH-12	94.306	91.689	91.397	88.889	63.929	80.212	85.897	80.057	69.722	93.831	71.958	80.857
CHI	94.220	90.270	92.500	89.306	63.020	80.156	87.179	79.345	69.588	94.675	58.413	82.143
IG	94.668	90.743	91.765	88.611	62.604	79.543	87.393	79.630	70.297	94.416	57.143	82.571
GR	94.234	91.554	90.809	90.139	64.995	80.435	86.859	78.917	69.779	94.870	57.884	83.714
GINI	94.566	89.662	91.397	88.333	62.604	77.871	87.821	78.632	70.067	94.286	57.460	82.000
JM	84.335	89.932	89.853	91.806	63.695	78.763	86.004	80.627	69.760	94.675	59.365	82.571
MDL	94.220	88.986	90.441	89.306	63.384	78.986	87.286	80.199	69.012	94.091	58.624	83.143
MGINI	94.046	89.932	90.662	86.806	61.980	77.982	85.577	75.214	69.741	94.351	54.815	79.143
RLV	94.436	91.351	91.471	87.917	62.578	79.376	87.927	80.057	69.530	94.545	60.847	82.000
RLF	76.257	87.365	86.985	82.778	61.746	72.018	81.624	80.342	68.782	87.597	65.503	74.143
SGAIN	86.156	92.635	91.985	88.333	65.203	80.992	87.607	80.627	70.297	94.805	57.778	82.714
SGINI	94.205	89.797	90.735	89.861	63.565	80.602	86.752	78.917	69.377	94.740	57.143	82.714
WOE	80.621	88.446	88.015	85.694	63.436	80.212	86.218	81.766	68.667	93.182	68.677	78.714

Table A.8 Hyper-heuristic using Attribute Value Count – Average Accuracy Results for Multiple Data Sets using 12 heuristics

A.3 Results for Hyper-heuristics that use Attribute Entropy

The hyper-heuristics in this section represent the problem state using the entropy of the attributes left. Each column represents pairs of values where the first value represents the average accuracy percentage value while the second value is the resultant p-value when comparing the results of the method to the results of the hyper-heuristic on the same data set.

	car		contrac		credit		derma		ecoli		flags	
HH-5	94.595	*p-value*	63.874	*p-value*	81.594	*p-value*	92.027	*p-value*	93.382	*p-value*	84.750	*p-value*
CHI	94.509	*0.872*	62.793	*0.416*	80.580	*0.543*	90.946	*0.421*	**94.706**	*0.199*	87.250	*0.288*
IG	**94.653**	*0.913*	62.432	*0.336*	77.391	*0.011*	92.297	*0.849*	93.088	*0.800*	87.250	*0.276*
GR	94.509	*0.873*	63.829	*0.969*	80.435	*0.411*	92.568	*0.733*	93.529	*0.904*	85.500	*0.734*
GINI	**94.653**	*0.913*	62.432	*0.345*	77.826	*0.008*	91.081	*0.506*	93.088	*0.800*	86.750	*0.404*
JM	84.191	*0.000*	63.243	*0.564*	81.232	*0.806*	91.351	*0.704*	92.206	*0.459*	86.250	*0.508*
MDL	94.509	0.873	63.829	*0.972*	80.870	*0.593*	90.000	*0.174*	92.941	*0.743*	85.750	*0.622*
MGINI	94.364	*0.656*	60.991	*0.030*	76.667	*0.003*	89.324	*0.117*	92.206	*0.322*	84.250	*0.811*
RLV	94.480	*0.822*	63.108	*0.590*	79.638	*0.207*	92.027	*1.000*	93.382	*1.000*	84.750	*1.000*
RLF	75.896	*0.000*	61.171	*0.030*	69.928	*0.000*	87.027	*0.003*	87.794	*0.002*	74.250	*0.000*
SGAIN	86.214	*0.000*	**64.234**	*0.775*	**82.101**	*0.752*	**92.973**	*0.523*	93.382	*1.000*	87.000	*0.293*
SGINI	94.480	*0.830*	63.559	*0.806*	79.855	*0.309*	90.541	*0.342*	93.235	*0.895*	**88.000**	*0.189*
WOE	80.289	*0.000*	63.468	*0.691*	81.522	*0.959*	90.135	*0.223*	90.294	*0.092*	82.250	*0.270*

	heart		ionosphere		spect		votes		wine		yeast	
HH-5	68.333	*p-value*	88.611	*p-value*	77.493	*p-value*	93.750	*p-value*	88.333	*p-value*	69.433	*p-value*
CHI	57.593	*0.000*	86.944	*0.281*	74.074	*0.351*	**93.864**	*0.917*	87.778	*0.790*	69.067	*0.795*
IG	58.519	*0.000*	88.333	*0.883*	75.783	*0.631*	92.273	*0.140*	88.611	*0.887*	69.555	*0.926*
GR	55.741	*0.000*	89.167	*0.709*	75.783	*0.570*	**93.864**	*0.910*	88.889	*0.780*	68.700	*0.628*
GINI	56.111	*0.000*	88.056	*0.719*	74.074	*0.341*	92.386	*0.189*	87.500	*0.686*	69.189	*0.847*
JM	63.889	*0.117*	87.083	*0.250*	78.917	*0.679*	92.386	*0.183*	87.500	*0.674*	68.517	*0.524*
MDL	57.407	*0.000*	88.750	*0.926*	76.638	*0.789*	92.614	*0.320*	88.889	*0.799*	69.738	*0.830*
MGINI	53.148	*0.000*	86.111	*0.164*	74.074	*0.221*	92.159	*0.128*	85.833	*0.260*	68.639	*0.574*
RLV	61.296	*0.008*	87.917	*0.662*	76.638	*0.813*	92.500	*0.226*	**89.444**	*0.590*	69.921	*0.710*
RLF	65.370	*0.312*	82.500	*0.000*	**78.917**	*0.654*	89.659	*0.003*	80.556	*0.002*	69.311	*0.919*
SGAIN	57.222	*0.000*	**89.306**	*0.639*	77.493	*1.000*	**93.864**	*0.905*	88.333	*1.000*	**69.982**	*0.701*
SGINI	56.667	*0.000*	87.500	*0.506*	74.359	*0.387*	93.068	*0.534*	89.167	*0.641*	69.616	*0.896*
WOE	**74.074**	*0.051*	88.472	*0.928*	78.917	*0.670*	93.182	*0.563*	88.333	*1.000*	69.189	*0.857*

Table A.9 Hyper-heuristic using Attribute Entropy – Average Accuracy Results for Single Data Sets using 5 heuristics

Tables A.9 and A.10 show results for hyper-heuristics trained and tested on single data sets. The hyper-heuristics in A.9 have a set of 5 heuristics at their disposal while the ones in A.10 utilize a set of 12 heuristics.

	car		contrac		credit		derma		ecoli		flags	
HH-12	94.624	*p-value*	63.551	*p-value*	80.725	*p-value*	93.378	*p-value*	93.088	*p-value*	81.250	*p-value*
CHI	94.711	*0.800*	62.275	*0.158*	80.797	*0.966*	91.486	*0.184*	92.500	*0.701*	83.750	*0.434*
IG	**94.942**	*0.334*	62.162	*0.122*	77.971	*0.108*	92.568	*0.519*	93.088	*1.000*	85.750	*0.079*
GR	94.798	*0.597*	**64.264**	*0.434*	80.580	*0.932*	91.081	*0.104*	90.588	*0.175*	**86.750**	*0.039*
GINI	94.884	*0.448*	62.125	*0.138*	77.246	*0.045*	91.216	*0.117*	93.088	*1.000*	84.500	*0.207*
JM	85.838	*0.000*	63.026	*0.537*	80.217	*0.749*	90.000	*0.009*	90.882	*0.141*	82.250	*0.694*
MDL	94.769	0.663	62.012	*0.143*	78.478	*0.202*	90.811	*0.040*	91.618	*0.381*	83.750	*0.382*
MGINI	94.624	*1.000*	61.449	*0.033*	76.812	*0.044*	89.730	*0.004*	**93.382**	*0.851*	81.250	*1.000*
RLV	94.827	*0.542*	62.462	*0.215*	79.275	*0.403*	91.892	*0.263*	93.235	*0.923*	84.000	*0.273*
RLF	76.387	*0.000*	60.998	*0.004*	68.406	*0.000*	86.351	*0.000*	88.088	*0.004*	76.250	*0.080*
SGAIN	86.879	*0.000*	64.039	*0.568*	**81.594**	*0.644*	**93.514**	*0.911*	92.647	*0.783*	83.500	*0.366*
SGINI	94.740	*0.740*	62.462	*0.192*	80.942	*0.903*	91.081	*0.074*	92.353	*0.638*	84.750	*0.222*
WOE	79.422	*0.000*	63.176	*0.668*	80.797	*0.968*	88.514	*0.004*	90.294	*0.095*	74.750	*0.016*

	heart		ionosphere		spect		votes		wine		yeast	
HH-12	**75.185**	*p-value*	87.778	*p-value*	79.101	*p-value*	92.273	*p-value*	87.778	*p-value*	70.226	*p-value*
CHI	56.111	*0.000*	**88.750**	*0.499*	78.042	*0.694*	92.614	*0.755*	88.889	*0.557*	69.738	*0.726*
IG	57.963	*0.000*	86.528	*0.466*	79.630	*0.849*	93.409	*0.372*	88.333	*0.794*	70.287	*0.961*
GR	56.667	*0.000*	87.778	*1.000*	78.571	*0.850*	**94.545**	*0.051*	88.889	*0.566*	70.165	*0.962*
GINI	54.630	*0.000*	86.528	*0.411*	77.778	*0.636*	93.523	*0.312*	**89.167**	*0.504*	70.775	*0.665*
JM	58.519	*0.000*	86.250	*0.317*	79.630	*0.864*	92.727	*0.715*	88.889	*0.656*	71.019	*0.538*
MDL	55.926	*0.000*	87.917	*0.930*	75.926	*0.220*	93.864	*0.195*	88.611	*0.688*	70.470	*0.837*
MGINI	51.667	*0.000*	85.694	*0.143*	75.926	*0.250*	93.182	*0.433*	87.222	*0.798*	**71.385**	*0.356*
RLV	60.000	*0.000*	88.611	*0.555*	78.571	*0.826*	92.500	*0.850*	86.667	*0.599*	70.775	*0.666*
RLF	63.148	*0.000*	80.833	*0.000*	78.307	*0.748*	86.250	*0.002*	82.778	*0.047*	70.043	*0.920*
SGAIN	58.519	*0.000*	87.639	*0.934*	**81.481**	*0.356*	93.864	*0.174*	88.056	*0.895*	70.226	*1.000*
SGINI	55.370	*0.000*	87.222	*0.719*	77.249	*0.516*	94.091	*0.123*	**89.167**	*0.504*	71.141	*0.478*
WOE	71.296	*0.189*	85.833	*0.237*	**81.481**	*0.392*	92.159	*0.924*	86.111	*0.477*	69.372	*0.521*

Table A.10 Hyper-heuristic using Attribute Entropy – Average Accuracy Results for Single Data Sets using 12 heuristics

Tables A.11 and A.12 show results for hyper-heuristics trained and tested on multiple data sets. The hyper-heuristics in A.11 have a set of 5 heuristics at their disposal while the ones in A.12 utilize a set of 12 heuristics.

	ca, de, ec, wi				co, cr, io, sp			
	car	derma	ecoli	wine	contrac	credit	ionosphere	spect
HH-5	94.509	92.243	91.441	87.667	63.919	80.087	86.722	74.556
CHI	94.358	90.703	92.500	88.500	63.027	79.594	86.611	72.481
IG	94.601	91.757	91.735	87.778	62.959	78.333	87.556	73.593
GR	94.445	91.622	90.559	88.778	63.959	79.522	87.389	74.556
GINI	94.462	90.757	91.618	87.444	63.101	77.594	87.806	72.481
JM	85.029	90.189	91.206	89.278	63.628	80.188	85.194	73.667
MDL	94.358	89.459	90.588	88.667	62.872	78.797	87.528	73.926
MGINI	94.191	89.703	91.147	87.944	61.486	76.913	85.500	72.111
RLV	94.457	91.703	91.676	88.500	63.209	79.043	87.583	73.333
RLF	76.653	86.568	87.294	81.444	61.635	69.594	81.139	75.704
SGAIN	86.306	93.162	92.382	87.778	64.547	81.768	87.139	74.370
SGINI	94.231	90.324	91.206	88.833	63.655	80.145	87.417	72.296
WOE	80.607	89.243	90.118	86.278	63.446	80.928	85.806	75.407

	ye, vo, he, fl			
	yeast	votes	heart	flags
HH-5	69.812	93.750	66.963	83.750
CHI	69.315	94.636	58.333	83.700
IG	69.530	93.773	59.037	84.950
GR	69.134	94.000	59.148	86.100
GINI	69.477	93.727	56.926	83.900
JM	69.275	93.864	62.593	82.650
MDL	69.403	93.682	59.741	83.850
MGINI	69.101	93.841	55.704	82.200
RLV	69.134	93.864	62.481	83.000
RLF	69.128	87.364	65.370	73.650
SGAIN	69.752	94.182	58.111	82.950
SGINI	69.154	94.023	57.593	84.500
WOE	68.752	92.864	69.926	78.350

Table A.11 Hyper-heuristic using Attribute Entropy – Average Accuracy Results for Multiple Data Sets using 5 heuristics

	ca, de, ec, wi				co, cr, io, sp			
	car	derma	ecoli	wine	contrac	credit	ionosphere	spect
HH	94.509	91.730	92.029	87.000	64.385	81.333	87.833	74.741
CHI	94.468	91.649	93.000	87.722	63.000	80.725	87.361	72.222
IG	94.659	92.486	92.324	88.278	62.959	78.638	88.250	72.667
GR	94.514	91.432	91.118	88.278	64.176	81.493	87.750	73.630
GINI	94.595	91.027	92.353	87.667	63.041	77.768	87.056	72.074
JM	84.988	90.865	91.765	89.222	64.304	81.043	85.583	73.296
MDL	94.497	90.243	91.294	87.333	62.709	80.203	88.139	72.852
MGINI	94.370	89.568	91.412	86.389	61.723	77.072	85.556	71.296
RLV	94.642	92.189	92.441	87.278	63.182	79.681	88.639	72.889
RLF	76.538	87.730	88.647	81.167	62.169	70.087	81.250	74.074
SGAIN	86.139	93.919	92.706	87.444	64.743	81.812	87.806	74.481
SGINI	94.428	91.162	91.676	88.222	63.608	80.493	87.944	72.222
WOE	80.942	89.459	90.676	85.833	64.277	81.696	85.250	74.370

	ye, vo, he, fl			
	yeast	votes	heart	flags
HH	69.678	93.750	70.185	83.500
CHI	68.852	94.386	57.667	85.400
IG	69.416	94.227	58.148	85.200
GR	68.423	94.500	57.852	86.150
GINI	69.161	94.273	56.333	84.200
JM	68.966	94.432	59.704	84.050
MDL	69.134	93.955	58.778	85.500
MGINI	68.940	94.273	55.259	84.400
RLV	69.094	94.250	59.630	83.600
RLF	68.711	87.773	64.333	75.000
SGAIN	69.456	94.705	58.185	85.000
SGINI	68.960	94.841	56.074	86.100
WOE	68.779	93.295	69.852	80.050

Table A.12 Hyper-heuristic using Attribute Entropy – Average Accuracy Results for Multiple Data Sets using 12 heuristics

A.4 Results for Hyper-heuristics that use Maximum Conditional Entropy

The hyper-heuristics in this section represent the problem state using the maximum conditional entropy of the class attribute. Each column represents pairs of values where the first value represents the average accuracy percentage value while the second value is the resultant p-value when comparing the results of the method to the results of the hyper-heuristic on the same data set.

	car		contrac		credit		derma		ecoli		flags	
HH-5	94.422	*p-value*	63.142	*p-value*	82.609	*p-value*	92.027	*p-value*	93.382	*p-value*	83.750	*p-value*
CHI	94.277	*0.709*	62.264	*0.343*	81.957	*0.623*	90.541	*0.271*	93.529	*0.925*	86.500	*0.220*
IG	**94.451**	*0.941*	61.892	*0.266*	78.116	*0.003*	91.081	*0.539*	93.971	*0.705*	86.750	*0.226*
GR	94.162	*0.531*	63.108	*0.972*	81.522	*0.448*	91.892	*0.917*	93.529	*0.924*	87.750	*0.114*
GINI	**94.451**	*0.941*	62.027	*0.324*	76.739	*0.001*	90.676	*0.383*	93.676	*0.848*	86.000	*0.311*
JM	85.202	*0.000*	62.635	*0.665*	82.174	*0.721*	90.135	*0.192*	94.118	*0.614*	84.000	*0.916*
MDL	94.220	0.618	62.128	*0.321*	82.174	*0.734*	89.324	*0.032*	92.647	*0.612*	87.000	*0.107*
MGINI	94.335	*0.821*	61.757	*0.204*	76.594	*0.000*	88.378	*0.010*	93.088	*0.850*	84.500	*0.756*
RLV	94.422	*1.000*	62.770	*0.710*	78.986	*0.014*	91.486	*0.726*	93.676	*0.848*	87.000	*0.107*
RLF	75.491	*0.000*	62.264	*0.372*	70.652	*0.000*	86.757	*0.001*	89.412	*0.015*	74.250	*0.001*
SGAIN	86.329	*0.000*	63.446	*0.769*	**83.696**	*0.446*	**93.378**	*0.336*	**94.265**	*0.606*	87.250	*0.103*
SGINI	94.191	*0.574*	**63.750**	*0.550*	79.275	*0.025*	90.000	*0.199*	93.088	*0.840*	**89.250**	*0.012*
WOE	78.092	*0.000*	62.770	*0.738*	81.087	*0.233*	89.459	*0.078*	91.029	*0.190*	78.750	*0.076*
	heart		ionosphere		spect		votes		wine		yeast	
HH-5	64.259	*p-value*	86.528	*p-value*	**81.852**	*p-value*	95.114	*p-value*	88.056	*p-value*	70.302	*p-value*
CHI	57.778	*0.021*	**88.194**	*0.340*	77.407	*0.011*	95.114	*1.000*	89.722	*0.481*	69.463	*0.394*
IG	56.481	*0.004*	85.417	*0.537*	78.519	*0.098*	94.659	*0.589*	89.167	*0.648*	70.067	*0.815*
GR	57.222	*0.014*	86.944	*0.793*	79.630	*0.247*	**95.568**	*0.567*	**91.389**	*0.139*	69.262	*0.314*
GINI	55.926	*0.002*	86.667	*0.932*	77.222	*0.007*	95.114	*1.000*	90.278	*0.306*	69.799	*0.612*
JM	59.444	*0.061*	85.417	*0.474*	78.889	*0.082*	95.000	*0.899*	90.556	*0.253*	69.295	*0.196*
MDL	56.296	*0.012*	86.667	*0.935*	77.593	*0.035*	95.114	*1.000*	90.556	*0.253*	69.362	*0.327*
MGINI	54.444	*0.001*	83.611	*0.161*	75.741	*0.001*	95.455	*0.700*	88.611	*0.817*	69.497	*0.447*
RLV	59.815	*0.158*	87.500	*0.583*	77.222	*0.015*	94.659	*0.637*	89.167	*0.604*	69.027	*0.220*
RLF	64.259	*1.000*	80.278	*0.003*	80.185	*0.346*	87.727	*0.000*	80.833	*0.004*	68.456	*0.036*
SGAIN	55.556	*0.001*	86.250	*0.864*	79.815	*0.302*	**95.568**	*0.567*	88.333	*0.876*	**70.570**	*0.775*
SGINI	54.259	*0.001*	85.556	*0.526*	77.778	*0.020*	95.114	*1.000*	90.000	*0.433*	70.101	*0.833*
WOE	**69.259**	*0.086*	85.833	*0.679*	80.556	*0.391*	94.318	*0.348*	85.000	*0.206*	68.289	*0.017*

Table A.13 Hyper-heuristic using Maximum Conditional Entropy – Average Accuracy Results for Single Data Sets using 5 heuristics

Tables A.13 and A.14 show results for hyper-heuristics trained and tested on single data sets. The hyper-heuristics in A.13 have a set of 5 heuristics at their disposal while the ones in A.14 utilize a set of 12 heuristics.

	car		contrac		credit		derma		ecoli		flags	
HH-12	**94.711**	*p-value*	63.716	*p-value*	81.232	*p-value*	92.703	*p-value*	92.206	*p-value*	80.250	*p-value*
CHI	94.162	*0.319*	62.466	*0.172*	80.942	*0.847*	91.081	*0.195*	**94.265**	*0.090*	82.750	*0.398*
IG	94.451	*0.639*	62.736	*0.336*	79.203	*0.183*	92.162	*0.690*	92.500	*0.808*	81.250	*0.739*
GR	94.191	*0.344*	64.257	*0.556*	80.000	*0.425*	89.595	*0.024*	92.500	*0.816*	83.000	*0.353*
GINI	94.422	*0.607*	63.209	*0.579*	78.478	*0.058*	91.351	*0.298*	92.353	*0.906*	79.250	*0.752*
JM	84.653	*0.000*	63.311	*0.687*	78.841	*0.183*	90.676	*0.096*	90.000	*0.131*	82.500	*0.439*
MDL	94.220	*0.371*	63.209	*0.579*	78.986	*0.165*	89.459	*0.015*	91.324	*0.540*	**84.500**	*0.177*
MGINI	94.046	*0.243*	61.892	*0.081*	77.174	*0.010*	89.865	*0.026*	92.794	*0.643*	78.750	*0.617*
RLV	94.566	*0.813*	62.196	*0.120*	79.783	*0.324*	92.027	*0.573*	92.353	*0.911*	81.000	*0.803*
RLF	75.607	*0.000*	61.216	*0.019*	70.580	*0.000*	87.297	*0.004*	86.912	*0.002*	77.250	*0.284*
SGAIN	86.185	*0.000*	**65.000**	*0.156*	81.304	*0.960*	**94.730**	*0.068*	92.206	*1.000*	83.750	*0.226*
SGINI	94.220	*0.375*	63.615	*0.915*	79.928	*0.383*	90.946	*0.131*	92.941	*0.550*	83.750	*0.222*
WOE	79.971	*0.000*	63.649	*0.951*	**83.261**	*0.200*	88.649	*0.002*	89.412	*0.116*	78.750	*0.612*
	heart		ionosphere		spect		votes		wine		yeast	
HH-12	**72.222**	*p-value*	84.167	*p-value*	80.556	*p-value*	92.841	*p-value*	86.667	*p-value*	69.396	*p-value*
CHI	56.481	*0.000*	87.083	*0.143*	78.519	*0.246*	**94.318**	*0.074*	87.222	*0.773*	70.403	*0.354*
IG	59.444	*0.000*	87.639	*0.055*	78.704	*0.299*	92.159	*0.445*	86.667	*1.000*	70.000	*0.566*
GR	58.148	*0.000*	**87.917**	*0.042*	77.963	*0.147*	93.409	*0.541*	**88.611**	*0.219*	70.067	*0.523*
GINI	57.037	*0.000*	86.389	*0.177*	78.333	*0.211*	92.045	*0.374*	85.833	*0.668*	70.201	*0.447*
JM	62.963	*0.008*	86.528	*0.196*	80.000	*0.726*	92.500	*0.718*	86.944	*0.886*	69.732	*0.735*
MDL	58.333	*0.000*	87.778	*0.067*	78.704	*0.275*	93.750	*0.352*	87.778	*0.495*	69.832	*0.710*
MGINI	55.000	*0.000*	83.750	*0.813*	75.185	*0.003*	92.386	*0.609*	87.500	*0.622*	69.832	*0.684*
RLV	60.741	*0.001*	86.389	*0.225*	79.259	*0.401*	92.841	*1.000*	86.667	*1.000*	69.597	*0.856*
RLF	67.222	*0.162*	80.972	*0.069*	79.630	*0.576*	86.477	*0.000*	83.056	*0.128*	68.960	*0.712*
SGAIN	55.926	*0.000*	87.778	*0.054*	**81.852**	*0.398*	93.523	*0.466*	88.333	*0.326*	**70.772**	*0.195*
SGINI	54.815	*0.000*	85.417	*0.515*	77.778	*0.106*	92.614	*0.800*	**88.611**	*0.219*	70.604	*0.275*
WOE	71.667	*0.881*	86.389	*0.242*	78.704	*0.222*	91.932	*0.369*	82.778	*0.068*	68.658	*0.533*

Table A.14 Hyper-heuristic using Maximum Conditional Entropy – Average Accuracy Results for Single Data Sets using 12 heuristics

Tables A.15 and A.16 show results for hyper-heuristics trained and tested on multiple data sets. The hyper-heuristics in A.15 have a set of 5 heuristics at their disposal while the ones in A.16 utilize a set of 12 heuristics.

	ca, de, ec, wi				co, cr, io, sp			
	car	derma	ecoli	wine	contrac	credit	ionosphere	spect
HH-5	94.277	90.946	93.088	88.056	64.324	80.507	84.722	77.778
CHI	94.509	90.000	95.147	87.778	63.243	82.101	87.083	75.741
IG	94.711	92.297	94.118	88.611	62.770	79.203	86.667	77.222
GR	94.538	89.324	93.529	88.889	63.649	81.087	85.694	76.296
GINI	94.653	89.595	93.971	87.500	62.973	79.348	86.944	75.741
JM	84.942	88.919	93.529	89.167	64.493	80.217	85.556	77.593
MDL	94.566	88.784	92.059	88.611	63.378	80.870	86.250	76.481
MGINI	94.364	88.784	93.382	86.389	62.061	77.536	86.528	74.444
RLV	94.711	91.757	94.412	88.333	62.872	79.203	85.694	76.852
RLF	76.705	87.297	88.088	82.222	62.466	72.464	80.139	77.778
SGAIN	86.214	93.243	94.853	87.500	63.818	81.739	85.278	77.963
SGINI	94.538	89.730	93.824	88.056	63.142	81.377	86.389	75.926
WOE	80.925	87.027	93.824	85.556	64.493	81.159	85.556	76.481

	ye, vo, he, fl			
	yeast	votes	heart	flags
HH-5	69.597	93.409	67.593	82.000
CHI	68.926	93.864	60.000	83.000
IG	68.523	93.750	56.296	83.500
GR	68.557	94.205	58.333	85.000
GINI	68.221	93.636	57.963	81.250
JM	68.725	93.295	62.037	82.750
MDL	68.490	93.750	60.370	85.500
MGINI	67.953	93.750	55.185	81.750
RLV	67.315	94.091	61.667	81.000
RLF	68.792	86.250	62.037	74.250
SGAIN	69.430	93.750	58.704	82.750
SGINI	68.658	93.750	59.815	86.250
WOE	68.356	93.750	69.815	75.000

Table A.15 Hyper-heuristic using Maximum Conditional Entropy – Average Accuracy Results for Multiple Data Sets using 5 heuristics

	ca, de, ec, wi				co, cr, io, sp			
	car	derma	ecoli	wine	contrac	credit	ionosphere	spect
HH-12	94.526	90.757	92.235	88.222	63.439	80.855	86.833	78.778
CHI	94.509	89.892	93.176	88.167	61.824	80.870	86.556	76.741
IG	94.757	91.243	92.941	88.444	62.034	78.580	87.306	76.630
GR	94.538	90.514	91.529	88.222	63.628	81.188	87.194	76.333
GINI	94.682	90.027	92.765	88.000	61.980	77.754	86.972	76.852
JM	84.665	89.622	91.529	90.167	63.486	80.420	85.583	77.778
MDL	94.526	88.649	91.618	87.611	62.358	80.362	87.972	76.778
MGINI	94.480	89.297	92.147	87.167	60.872	76.594	85.639	74.111
RLV	94.584	90.676	92.882	87.500	62.324	79.203	87.333	76.519
RLF	76.272	86.703	88.147	80.389	61.162	69.986	80.250	78.333
SGAIN	86.058	92.973	93.412	88.333	63.838	81.493	87.472	78.444
SGINI	94.491	89.892	92.118	88.889	62.696	80.130	87.417	76.815
WOE	80.624	88.730	90.882	86.667	63.405	81.101	85.889	78.519

	ye, vo, he, fl			
	yeast	votes	heart	flags
HH-12	69.523	93.614	70.074	80.300
CHI	69.416	94.614	58.778	82.750
IG	69.812	94.341	58.259	82.450
GR	68.886	94.795	57.926	83.600
GINI	69.530	94.477	57.815	81.550
JM	69.275	93.773	60.630	81.800
MDL	69.591	94.000	59.259	82.400
MGINI	69.745	94.159	56.074	80.700
RLV	69.523	94.182	60.148	81.200
RLF	68.678	86.591	64.000	75.150
SGAIN	69.664	94.545	57.630	82.500
SGINI	69.497	94.932	57.148	83.850
WOE	69.168	93.114	71.593	78.400

Table A.16 Hyper-heuristic using Maximum Conditional Entropy – Average Accuracy Results for Multiple Data Sets using 12 heuristics

References

Abe H. and Yamaguchi T. (2004) *Constructive meta-learning with machine learning method repositories.* In Proceedings of the 17th international conference on Innovations in applied artificial intelligence, pp. 502-511.

Abellán J. and Moral S. (2003) *Building classification trees using the total uncertainty criterion.* In International Journal of Intelligent Systems, vol. 18, no. 12, pp. 1215-1225.

Abellán J. and Moral S (2005) *Upper entropy of credal sets. Applications to credal classification.* In International Journal of Approximate Reasoning, vol. 39, no. 2-3, pp. 235-255.

Ahmadi S., Barrone P., Cheng P., Burke E. K., Cowling P. and McCollum B. (2003) *Perturbation based variable neighbourhood search in heuristic space for examination timetabling problem.* In Proceedings of Multidisciplinary International Scheduling: Theory and Applications, Nottingham, August 13-16, pp. 155-171.

Asmuni H., Burke E. K. and Garibaldi J. M. (2005) *Fuzzy multiple heuristic ordering for course timetabling.* In Proceedings of the 5th United KingdomWorkshop on Computational Intelligence, London, UK, pp. 302-309.

Asuncion A. and Newman D. J. (2009) *UCI Machine Learning Repository*, available online: http://www.ics.uci.edu/~mlearn/MLRepository.html. University of California, School of Information and Computer Science, Irvine, California, US.

Ayob M. and Kendall G. (2003) *A Monte Carlo Hyper-Heuristic To Optimise Component Placement Sequencing For Multi Head Placement Machine.* In Proceedings of the International Conference on Intelligent Technologies, pp. 132-141.

Azam M., Zaman Q. and Pfeiffer K. P. (2007) *Improved classification trees with two or more classes.* In Proceedings of the 9th Islamic Countries Conference on Statistical Sciences ICCS-IX, pp. 608-626.

Bader-El-Din MB, Poli R (2007) Generating sat local-search heuristics using a gp hyper-heuristic framework. In: LNCS 4926. Proceedings of the 8th International Conference on Artifcial Evolution, pp 37-49.

Badulescu L.A. (2007) *The Choice of the Best Attribute Selection Measure in Decision Tree Induction.* Annals of University of Craiova, Mathematics and Computer Science Series, vol. 34, pp. 88-93.

Bai R. and Kendall G. (2005) *An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics.* In Metaheuristics: Progress as Real Problem Solvers, Operations Research/Computer Science Interfaces, vol. 32, Springer, pp. 87-108.

Bai R., Burke E. K., Kendall G. and McCollum B. (2006) *A Simulated annealing Hyper-Heuristic for University Course Timetabling Problem.* In Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, Brno, pp. 345-350.

Bai R., Blazewicz J., Burke E. K., Kendall G. and McCollum B. (2007a) *A simulated annealing hyper-heuristic methodology for flexible decision support.* Technical report, School of Computer Science, University of Nottingham, UK.

Bai R., Burke E. K., Kendall G. and McCollum B. (2007b) *Memory Length in Hyper-heuristics: An Empirical Study.* In Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling, pp. 173-178.

Bai R., Burke E. K. and Kendall G. (2008) *Heuristic, Meta-heuristic and Hyper-heuristic Approaches for Fresh Produce Inventory Control and Shelf Space Allocation.* In Journal of the Operational Research Society, vol. 59 (10), pp. 187-1397.

Baim P. W. (1988) *A Method for Attribute Selection in Inductive Learning Systems.* In IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 10, no. 6, pp. 888-896.

Baker E. and Jain A. K. (1976) *On feature ordering in practice and some finite sample effects.* In Proceedings of the Third International Joint Conference on Pattern Recognition, San Diego, California, pp. 45-49.

Barros R. C., Carvalho A. C. P. L. F., Basgalupp M. P. and Freitas A. A. (2011) *Towards the automatic design of decision tree induction algorithms.* In Proceedings of the GECCO-2011 First Workshop on Evolutionary Algorithms for Evolving Generic Algorithms, pp. 567-574.

Ben-Bassat M. (1978) *Myopic policies in sequential classification.* In IEEE Transacions on Computing, vol. 27 (2), pp.170-174.

Mary Saira Bhanu S. and Gopalan N. P. (2008) *A Hyper-Heuristic approach for Efficient Resource Scheduling in Grid.* In International Journal of Computers, Communications and Control, vol. III, no. 3, pp. 249-258.

Biazzini M., Banhelyi B., Montresor A., Jelasity M. (2009) *Distributed hyper-heuristics for real parameter optimization.* In Proceedings of the 11th Annual

Conference on Genetic and Evolutionary Computation, Montreal, Québec, Canada, pp.1339-1346.

Bilgin B., Ozcan E., and Korkmaz E. E. (2006) *An experimental study on hyper-heuristics and final exam scheduling.* In Proceedings of the International Conference on the Practice and Theory of Automated Timetabling, pp. 123-140.

Breiman L., Friedman J., Olshen R., and Stone C. (1984) *Classification and Regression Trees.* Wadsworth International Group.

Breiman L. (1996) *Technical note: some properties of splitting criteria.* In Machine Learning, vol. 24, no.1, pp. 41-47.

Buntine W. and Niblett T. (1992) *A further comparison of splitting rules for decision-tree induction.* In Machine Learning, vol. 8, pp. 75-85.

Burke E. K. and Newall J. P. (2002) *A New Adaptive Heuristic Framework for Examination Timetabling Problems.* In Technical Report NOTTCS-TR-2001-5, University of Nottingham, UK, School of Computer Science & I.T.

Burke E. K., MacCarthy B., Petrovic S., and Qu R. (2002) *Knowledge discovery in hyper-heuristics using case-based reasoning on course timetabling.* In Full Proceedings of the Fourth International Conference on the Practice And Theory of Automated Timetabling, pp. 90–103.

Burke E. K., Hart E., Kendall G., Newall J., Ross P. and Schulenburg S. (2003a) *Hyper-heuristics: An emerging direction in modern search technology.* In Handbook of Metaheuristics, Kluwer, pp. 457–474.

Burke E. K., Kendall G. and Soubeiga E. (2003b) *A Tabu-Search Hyperheuristic for Timetabling and Rostering.* In Journal of Heuristics, vol. 9, no. 6, pp. 451-470.

Burke E. K., Silva J. D. L. and Soubeiga E. (2003c) *Hyperheuristic Approaches for Multiobjective Optimisation.* In 5th Metaheuristics International Conference, pp. 11.1--11.6.

Burke E. K., Landa-Silva J. D. and Soubeiga E. (2003d) *Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling.* In Meta-heuristics: Progress as Real Problem Solvers, Selected Papers from the 5th Metaheuristics International Conference, Springer, pp. 129-158.

Burke E. K., Dror M., Petrovic S. and Qu R. (2005a) *Hybrid Graph Heuristics within a Hyper-heuristic Approach to Exam Timetabling Problems.* The Next Wave in Computing, Optimization, and Decision Technologies, Springer.

Burke E. K., Kendall G., Landa-Silva J. D., O'Brien R. and Soubeiga E. (2005b) *An ant algorithm hyperheuristic for the project presentation scheduling problem.* In Proceedings of the 2005 IEEE Congress on Evolutionary Computation, Edinburgh, Scotland, vol. 3, pp 2263-2270.

Burke E. K., Hyde M. and Kendall G. (2006a) *Evolving Bin Packing Heuristics with Genetic Programming.* In Proceedings of the 9th International Conference on Parallel Problem Solving from Nature, Reykjavik, Iceland, pp. 9-13.

Burke E. K., Petrovic S. and Qu R. (2006b) *Case Based Heuristic Selection for Timetabling Problems.* In Journal of Scheduling, vol. 9 (2), pp. 115-132.

Burke E. K., McCollum B., Meisels A., Petrovic S. and Qu R. (2007a), *A Graph-Based Hyper-Heuristic for Timetabling Problems.* In European Journal of Operational Research, vol. 176, no. 1, pp 177-192.

Burke E. K., Hyde M. R., Kendall G. and Woodward J. (2007b) *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, July 07-11, 2007.

Burke E. K., Hyde M. R., Kendall G. and Woodward J. R. (2007c) *The Scalability of Evolved On Line Bin Packing Heuristics.* In Proceedings of 2007 IEEE Congress on Evolutionary Computation, pp. 2530-2537.

Burke E. K. and Bykov Y. (2008) *A late acceptance strategy in hill-climbing for exam timetabling problems.* In the 7th International Conference on the Practice and Theory of Automated Timetabling, Montréal, August 18-22, 2008.

Burke E. K., Kendall G., Özcan E. (2008a) *Monte carlo hyper-heuristics for examination timetabling.* In the 7th International Conference on the Practice and Theory of Automated Timetabling, Montréal, Canada, August 18-22, 2008.

Burke E. K., Misir M., Ochoa G. and Özcan E. (2008b) *Learning Heuristic Selection in Hyperheuristics for Examination Timetabling.* In the 7th International Conference on the Practice and Theory of Automated Timetabling, Montréal, Canada, August 18-22, 2008.

Burke E. K., Li J. and Qu R. (2008c) *Data Mining: an Aid Towards more Efficient Hyper-heuristic Search.* In the 7th International Conference on the

Practice and Theory of Automated Timetabling, Montréal, Canada, August 18-22, 2008.

Burke E. K., Hyde M. R., Kendall G., and Woodward J. (2008d) *A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics.* Technical Report, University of Nottingham, Department of Computer Science.

Burke E. K., Kendall G., Misir M. and Özcan E. (2008e) *A Study of Simulated Annealing Hyperheuristics.* In the 7th International Conference on the Practice and Theory of Automated Timetabling, Montréal, Canada, August 18-22, 2008.

Burke E. K., Hyde M. R., Kendall G., Ochoa G., Ozcan E. and Woodward J. R. (2009a) *Exploring Hyper-heuristic Methodologies with Genetic Programming.*Computational Intelligence: Collaboration, Fusion and Emergence, Intelligent Systems Reference Library, Springer, pp. 177-201.

Burke E. K., Hyde M., Kendall G., Ochoa G., Ozcan E., and Woodward J. R. (2009b) *A classification of hyper-heuristic approaches.* Technical Report No: NOTTCS-TR-SUB-0907061259-5808, Univeristy of Nottingham, UK.

Burke E. K., Hyde M., Kendall G., Ochoa G., Ozcan E., and Qu R. (2009c) *A survey of hyper-heuristics*, Technical Report No: NOTTCS-TR-SUB-0906241418-2747, Univeristy of Nottingham.

Casey R. G. and Nagy G. (1984) *Decision tree design using a probabilistic model.* In IEEE Transactions on Information Theory, IT-30 (1), pp. 93-99.

Cestnik B., Kononenko I. and Bratko I. (1987) *ASSISTANT 86: A Knowledge Elicitation Tool for Sophistical Users.* In Proceedings of the 2nd European Working Session on Learning, pp.31-45.

Chakhlevitch K. and Cowling P. (2005). *Choosing the Fittest Subset of Low Level Heuristics in a Hyperheuristic Framework.* In Proceedings of 5th European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, Lecture Notes in Computer Science, vol. 3448, pp 25-33.

Chandra B., Kothari R. and Paul P. (2010) *A new node splitting measure for decision tree construction.* Pattern Recognition Archive, vol. 43, no. 8, pp. 2725-2731.

Chen P. C., Kendall G. and Vanden-Berghe G. (2007) *An ant based hyper-heuristic for the travelling tournament problem.* In Proceedings of IEEE Symposium of Computational Intelligence in Scheduling, Hawaii, pp. 19-26.

Cieslak D. A. and Chawla N. V. (2008) *Learning Decision Trees for Unbalanced Data.* In European Conference on Machine Learning, pp. 241-256.

Corne D. and Ross P. (1995) *Peckish Initialisation Strategies for Evolutionary Timetabling.* In Practice and Theory of Automated Timetabling , Springer Lecture Notes in Computer Science no. 1153, pp. 227-240.

Corne D. and Ogden J. (1997) *Evolutionary optimisation of methodist preaching timetables.* In Second International Conference on the Practice And Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1408, pp. 142-155.

Cowling P., Kendall G. and Soubeiga E. (2000) *A hyperheuristic approach for scheduling a sales summit.* In Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, Lecture Notes in Computer Science, pp. 176–190.

Cowling P., Kendall G. and Soubeiga E. (2001) *A parameter-free hyperheuristic for scheduling a sales summit.* In Proceedings of the 4th Metaheuristic International Conference, pp. 127-131.

Cowling P., Kendall G. and Soubeiga E. (2002a) *Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation.* In Second European Conference on Evolutionary Computing for Combinatorial Optimisation, Lecture Notes in Computer Science, pp. 1–10.

Cowling P., Kendall G. and Han L. (2002b) *An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem.* In Congress on Evolutionary Computation, pp. 1185–1190.

Cowling P., Kendall G. and Han L. (2002c) *An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem.* In Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning, Orchid Country Club, Singapore, pp. 267-271.

Cowling, P. and Chakhlevitch K. (2003) *Hyperheuristics for managing a large collection of low level heuristics to schedule personnel.* In Proceedings of the 2003 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1214–1221.

Crowston W. B., Glover F., Thompson G. L. and Trawick J. D. (1963) *Probabilistic and parametric learning combinations of local job shop scheduling rules.* Office of Naval Research, Research Memorandum, GSIA, Carnegie Mellon University, Pittsburgh, US.

Delibasic B., Jovanovic M., Vukicevic M., Suknovic M. and Obradovic Z. (2011) *Component-based decision trees for classification.* In Intelligent Data Analysis, Vol. 15, Nr. 5, pp. 671-693.

Dietterich T., Kearns M. and Mansour Y. (1996) *Applying the weak learning framework to understand and improve C4.5.* In Proceedings of the 13th International Conference on Machine Learning, Morgan Kaufmann, San Francisco, pp. 96-104.

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

Domingos P. (1998) *Occam's Two Razors: The Sharp and the Blunt.* In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, pp. 37-43.

Dorigo M., Maniezzo V. and Colorni A. (1991) *The ant system: an autocatalytic optimizing process.* Technical Report TR91-016, Politecnico di Milano, Italy.

Dorndorf U. and Pesch E. (1995) *Evolution based learning in a job shop scheduling environment.* Computers and Operations Research, vol. 22(1), pp. 25-40.

Dowsland K. A., Soubeiga E., and Burke E. K. (2007) *A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation.* In European Journal of Operational Research, vol. 179, no. 3, pp. 759-774.

Ersoy E., Ozcan E., Etaner-Uyar A. S. (2007) *Memetic algorithms and hyperhill-climbers.* In the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications, pp 159-166.

Esmeir S. and Markovitch S. (2006) *When a decision tree learner has plenty of time.* In Proceedings of the 21st National Conference on Artificial Intelligence, Boston, Massachusetts, pp.1597-1600.

Fang H. L., Ross P. and Corne D. (1993) *A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems.* In Fifth International Conference on Genetic Algorithms, San Mateo, Morgan Kaufmann, pp. 375-382.

Fang H. L., Ross P. and Corne D. (1994) *A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems.* In Proceedings of the 11th European Conference on Artificial Intelligence, pp. 590-594.

Fayyad U. M. and Irani K. B. (1993) *Multi-interval Discretization of Continuous-Valued Attributes for Classification Learning.* In Proceedings of 13th International Joint Conference on Artificial Intelligence, pp. 1022-1027.

Fayyad U. M. and Irani K. B. (1992) *The attribute specification problem in decision tree generation.* In Proceedings of AAAI-92, San Jose CA. AAAI Press, pp. 104-110.

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

Fleurent C. and Ferland J. A. (1996) *Genetic and hybrid algorithms for graph coloring.* Annals of Operations Research, vol. 63, pp. 437-461.

Freitas A. A. (2002) *Data Mining and Knowledge Discovery with Evolutionary Algorithms.* Springer-Verlag, Berlin, Germany.

Friedman J. H. (1977) *A recursive partitioning decision rule for nonparametric classifers.* In IEEE Transactions on Computation, C-26, pp. 404-408.

Fukunaga A. (2002) *Automated discovery of composite SAT variable-selection heuristics.* In the Eighteenth National Conference on Artificial intelligence, Edmonton, Alberta, Canada, pp. 641-648.

Fukunaga A. (2004) *Evolving local search heuristics for SAT using genetic programming.* In Genetic and Evolutionary Computation Conference 2004, part II, pp. 483–494.

Fukunaga A. (2008) *Automated discovery of local search heuristics for satisfiability testing.* Evolutionary Computation, vol.16, no.1, pp.31-61.

Garrido P. and Riff M. C. (2007a) *Collaboration between hyperheuristics to solve strippacking problems.* In Proceedings of 12th International Fuzzy Systems Association World Congress, Springer, Lecture Notes in Computer Science, vol. 4529, pp. 698-707.

Garrido P., Riff M. C. (2007b) *An evolutionary hyperheuristic to solve strip-packing problems.* In Proceedings of Intelligent Data Engineering and Automated Learning, Springer, Lecture Notes in Computer Science, vol. 4881, pp. 406-415.

Gaw A., Rattadilok P. and Kwan R. S. K. (2004) *Distributed Choice Function Hyperheuristics for Timetabling and Scheduling.* In Proceedings of the 5th

International Conference on the Practice and Theory of Automated Timetabling, 2004, pp. 495-498.

Geiger C. D, Uzsoy R. and Aytug H. (2006) *Rapid Modeling and Discovery of Priority Dispatching Rules: An Autonomous Learning Approach.* In Journal of Scheduling , vol. 9, no. 1, pp. 7-34.

Gelfand S. B., Ravishankar C. S. and Delp E. J. (1991) *An iterative growing and pruning algorithm for classifcation tree design.* In IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 13, no. 2, pp. 163-174.

Gillo M. W. (1972) *MAID: A Honeywell 600 program for an automatised survey analysis.* Behavioral Science, vol. 17, pp. 251-252.

Glesser M. A. and Collen M. F. (1972) *Towards automated medical decisions.* Comp. and Biomedical Research, vol. 5, no. 2, pp. 180-189.

Gratch J., Chien S. and DeJong G. (1993) *Learning search control knowledge for deep space network scheduling.* In Proceedings of 10th International Conference on Machine Learning, Amherst, MA, pp. 135–142.

Gratch J. and Chien S. (1996) *Adaptive problem-solving for large-scale scheduling problems: a case study.* Journal of Artificial Intelligence Research, vol. 4, pp. 365-396.

Grewe L. and Kak A. C. (1995) *Interactive learning of a multi-attribute hash table classifier for fast object recognition.* Computer Vision and Image Understanding, vol. 61, no. 3, pp 387-416.

Han J. and Kamber M. (2000) *Data mining: concepts and techniques.* Morgan Kaufmann, San Francisco, CA, US.

Han L. and Kendall G. (2003) *Guided operators for a hyper-heuristic genetic algorithm.* In Proceedings of AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference on Artificial Intelligence, Perth, Australia, pp. 807-820.

Hanisch W. (1990) *Design and optimization of a hierarchical classifer.* Journal of New Generation Computer Systems, vol. 3, no. 2, pp. 159-173.

Hart A. (1984) *Experience in the use of an inductive system in knowledge engineering.* In Research and Development in Expert Systems. Cambridge University Press, Cambridge, MA.

Hart E. and Ross P. (1998) *A heuristic combination method for solving job-shop scheduling problems.* In Parallel Problem Solving from Nature, Springer-Verlag, Lecture Notes in Computer Science, vol. 1498, pp. 845-854.

Hart E., Ross P. and Nelson J. A. D. (1998) *Solving a real-world problem using an evolving heuristically driven schedule builder.* Evolutionary Computing, vol. 6, no. 1, pp. 61-80.

Hartmann C., Varshney P., Mehrotra K. and Gerberich C. (1982) *Application of information theory to the construction of efficient decision trees.* In IEEE Transactions on Information Theory, vol. 28, no. 4, pp. 565-577.

Haskell R. E. and Noui-Mehidi A. (1989) *Design of hierarchical classifers.* In Computing in the 90's: The First Great Lakes Computer Science Conference Proceedings, pp. 118-124.

Heath D., Kasif S. and Salzberg S. (1993) *Learning oblique decision trees.* In International Joint Conference on Artificial Intelligence, pp. 1002-1007.

Ho N. B. and Tay J. C. (2005) *Evolving dispatching rules for solving the flexible job-shop problem.* In Proceedings of the IEEE Congress on Evolutionary Computation, Edinburgh, UK, pp. 2848-2855.

Holland J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, University of Michigan Press.

Jakobovic D., Jelenkovic L. and Budin L. (2007) *Genetic programming heuristics for multiple machine scheduling.* In Proceedings of the European Conference on Genetic Programming, Valencia, Spain, pp. 321-330.

Johnson D., Demers A., Ullman J., Garey M. and Graham R. (1974) *Worst-case performance bounds for simple one-dimensional packaging algorithms.* SIAM Journal on Computing 3, pp. 299-325.

Jones T. (1995) *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD Thesis, The University of New Mexico, U.S.

Joslin D. E. and Clements D. P. (1999) *Squeaky wheel optimisation.* Journal of Artificial Intelligence, vol. 10, pp. 353-373.

Kalkanis G. (1993) *The application of confidence interval error analysis to the design of decision tree classifiers.* Pattern Recognition Letters, vol. 14, no. 5, pp. 355-361.

Kass G. V. (1980) *An exploratory technique for investigating large quantities of categorical data.* Applied Statistics, vol. 29, no. 2, pp. 119-127.

Keller R. E. and Poli R. (2007a) *Cost-benefit investigation of a genetic-programming hyperheuristic.* In Proceedings of the 8th International Conference on Artifcial Evolution, Tours, France, pp. 13-24.

Keller R. E. and Poli R. (2007b) *Linear genetic programming of parsimonious metaheuristics.* In Proceedings of the IEEE Congress on Evolutionary Computation, Singapore, pp. 4508-4515.

Keller R. E. and Poli R. (2008a) *Self-adaptive hyperheuristic and greedy search.* In 2008 IEEE World Congress on Computational Intelligence, Hong Kong, pp. 3801-3808.

Keller R. E. and Poli R. (2008b) *Subheuristic search and scalability in a hyperheuristic.* In Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, pp. 609-610.

Keller R. E. and Poli R. (2008c) *Toward subheuristic search.* In IEEE World Congress on Computational Intelligence, Hong Kong, pp. 3148-3155.

Kelly J. D. and Davis L. (1991) *Hybridizing the genetic algorithm and the k nearest neighbors classification algorithm.* In Fourth International Conference on Genetic Algorithms, pp. 377-383.

Kendall G., Soubeiga E. and Cowling P. (2002a) *Hyperheuristics: a robust optimisation method for real-world scheduling.* In Proceedings of 7th International Conference on Parallel Problem Solving from Nature.

Kendall G., Soubeiga E. and Cowling P. (2002b) *Choice function and random hyperheuristics.* In 4th Asia-Pacific Conference on Simulated Evolution And Learning, pp. 667–671.

Kendall G. and Mohamad M. (2004a) *Channel Assignment in Cellular Communication Using a Great Deluge Hyper-heuristic.* In Proceedings of the 2004 12th IEEE International conference on Networks, Singapore, pp. 769-773.

Kendall G. and Mohamad M. (2004b) *Channel assignment optimisation using a hyperheuristic.* In Proceedings of the 2004 IEEE Conference on Cybernetic and Intelligent Systems, Singapore, pp. 790-795.

Kendall G. and Hussin N. M. (2005) *An investigation of a tabu search based hyper-heuristic for examination timetabling.* In Multidisciplinary Scheduling: Theory and Applications, Springer, pp. 309-328.

Kerber R. (1992) *ChiMerge: Discretization of Numeric Attributes.* In Proceedings of 9th International Conference of AAAI, pp. 123-128.

Kim H. and Loh W. (2001) *Classification trees with unbiased multiway splits.* Journal of the American Statistical Association, vol. 96, pp. 589-604.

Kim H. and Loh W. (2003) *Classification trees with bivariate linear discriminant node models.* Journal of Computational and Graphical Statistics, vol. 12, pp. 512-530.

Kira K. and Rendell L. (1992). *The Feature Selection Problem: traditional methods and a new algorithm.* In Proceedings of the 10th National Conference on Artificial Intelligence, pp. 129-134.

Kononenko I. (1994). *Estimating Attributes: Analysis and Extensions of RELIEF.* In Proceedings of 7th European Conference on Machine Learning, pp. 171-182.

Kononenko I. (1995). *On Biases in Estimating Multi-Valued Attributes.* In Proceedings of 1st International Conference on Knowledge Discovery and Data Mining, pp. 1034–1040.

Kumar R., Joshi A. H., Banka K. K. and Rockett P. (2008) *Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming.* In Proceedings of the 10th ACM conference on Genetic and Evolutionary Computation, Atlanta, GA, USA, pp. 1227-1234.

Kurzynski M. W. (1983) *The optimal strategy of a tree classifier.* Pattern Recognition, vol. 16, pp. 81-87.

Kurzynski M. W. (1988) *On the multi-stage Bayes classifier.* Pattern Recognition Letters, vol. 21, no. 4, pp. 355-365.

Kurzynski M. W. (1989) *On the identity of optimal strategies for multi-stage classifiers.* Pattern Recognition Letters, vol. 10, no. 1, pp. 39-46.

Langdon W. B. (1995) *Scheduling planned maintenance of the national grid.* In Evolutionary Computing, no. 993 in Lecture Notes in Computer Science, pp. 132-153.

Langon W. B. (1996) *Scheduling maintenance of eletrical power transmission networks using genetic programming.* In Late-breaking papers, Genetic Programming Conference.

Li X. and Dubes R. C. (1986) *Tree classifier design with a permutation statistic.* Pattern Recognition, vol. 19, no. 3, pp. 229-235.

Lim T., Loh W. and Shih Y. (2000) *A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms*. Machine Learning, vol. 40, no. 3, pp. 203-228.

Lin Y. K. and Fu K. (1983) *Automatic classifcation of cervical cells using a binary tree classifer*. Pattern Recognition, vol. 16, no. 1, pp. 69-80.

Ling S. E. (1992) *Integrating genetic algorithms with a prolog assignment program as a hybrid solution for a polytechnique timetable problem*. In Parallel Problem Solving from Nature, vol. 2, pp. 321-329.

Liu W. Z. and White A. P. *The importance of attribute selection measures in decision tree induction*. Machine Learning, vol. 15, pp. 25-41.

Loh W. and Vanichsetakul N. (1988), *Tree-structured classification via generalized discriminant analysis*. Journal of the American Statistical Association, vol. 83, pp. 715-728.

Loh W. and Shih Y. (1997) *Split selection methods for classification trees*. Statistica Sinica, vol. 7, pp. 815-840.

Loh W. (2009) *Improving the precision of classification trees*. DOI Annals of Applied Statistics, vol. 3, pp. 1710-1737.

López De Mántaras R. (1991). *A Distance-Based Attribute Selection Measure for Decision Tree Induction*. Machine Learning vol. 6, pp. 81-92.

Lubinsky D. (1993) *Algorithmic speedups in growing classification trees by using an additive split criterion*. In AI & Statistics-93, pp. 435-444.

Lubinsky D. (1994) *Bivariate splits and consistent split criteria in dichotomous classification trees.* PhD thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1994.

Luo R. C., Scherp R. S. and Lanzo M. (1987) *Object identification using automated decision tree construction approachfor robotics applications.* Journal of Robotic Systems, vol. 4, no. 3, pp. 423-433.

Marín-Blázquez J. G. and Schulenburg S. (2006) *Multi-step environment learning classifier systems applied to hyper-heuristics.* In Proceedings of the 8th annual Conference on Genetic and Evolutionary Computation, pp. 1521-1528.

Marín-Blázquez J. G. and Schulenburg S. (2007) *A hyper-heuristic framework with XCS: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients.* In IWLCS, Springer, Lecture Notes in Computer Science, vol. 4399, pp 193-218.

Marques de Sá J., Gama J., Sebastião R. and Alexandre L. A. (2009) *Decision Trees Using the Minimum Entropy-of-Error Principle.* In Proceedings of CAIP 2009, pp. 799-807.

Mehta M., Rissanen J. and Agrawal R. (1995) *MDL-based decision tree pruning.* In Proceedings of the First international Conference on Knowledge Discovery and Data Mining, AAAI Press, pp. 216-221.

Van de Merckt T. (1993) *Decision trees in numerical attribute spaces.* In International Joint Conference on Artificial Intelligence, pp. 1016-1021.

Michie D. (1989). *Personal Models of Rationality.* Journal of Statistical Planning and Inference, Special Issue on Foundations and Philosophy of Probability and Statistics.

Mingers J. (1987) *Expert systems - rule induction with statistical data.* Journal of the Operational Research Society, vol. 38, no. 1, pp. 39-47.

Mingers J. (1989) *An Empirical Comparison of Selection Measures for Decision-Tree Induction.* Machine Learning, vol. 3, no. 4, pp. 319-342.

Minton S. (1996) *Automatically configuring constraint satisfaction problems: a case study.* Constraints 1 (1), pp. 7-43.

Miyakawa M. (1989) *Criteria for selecting a variable in the construction of efficient decision trees.* In IEEE Transactions on Computation, vol. 38, no. 1, pp. 130-141.

Moret B. M. E., Thomason M. G. and Gonzalez R. C. (1980) *The activity of a variable and its relation to decision trees.* ACM Transactions on Programming Language Systems, vol. 2, no. 4, pp. 580-595.

Morgan J. N. and Sondquist J. A. (1963) *Problems in the analysis of survey data, and a proposal.* J. Amer. Statist. Assoc., 58, pp. 415-434.

Morgan J. N. and Messenger R. C. (1973) *THAID: a sequential search program for the analysis of nominal scale dependent variables.* Technical report, Institute for Social Research, University. of Michigan, Ann Arbor, MI, US.

Murthy S. K., Kasif S., Salzberg S. and Beigel R. (1993) *OC1: Randomized induction of oblique decision trees.* In Proceedings of National Conferenceon Artificial Intelligence, pp. 322-327.

Murthy S. K., Kasif S. and Salzberg S. (1994) *A system for induction of oblique decision trees.* Journal of Artificial Intelligence Research, vol. 2, pp. 1-33.

Nareyek A. (2001) *Choosing search heuristics by non-stationary reinforcement learning.* In Metaheuristic International Conference, Porto, Portugal.

Norenkov I. and Goodman E. (1997) *Solving scheduling problems via evolutionary methods for rule sequence optimization.* In Proceedings of 2nd World Conference on Soft Computing, WSC2.

Norton S. W. (1989) *Generating better decision trees.* In Proceedings of the 11th International Joint Conference on Artificial Intelligence, vol. 1, pp. 800-805.

Ochoa G., Vázquez-Rodríguez J. A., Petrovic S., Burke E. K. (2009a) *Dispatching rules for production scheduling: a hyper-heuristic landscape analysis.* In Proceedings of the IEEE Congress on Evolutionary Computation, Montreal, Norway.

Ochoa G., Qu R. and Burke E. K. (2009b) *Analyzing the Landscape of a Graph Based Hyper-heuristic for Timetabling Problems.* The Genetic and Evolutionary Computation Conference, pp. 341-348.

Oltean M. and Dumitrescu D. (2004) *Evolving TSP heuristics using multi expression programming.* In Proceedings of the 4th International Conference on Computational Science, Krakow, Poland, pp. 670-673.

Oltean M. (2005) *Evolving evolutionary algorithms using linear genetic programming.* Evolutionary Computation, vol. 13, no. 3, pp. 387-410.

Ong Y. S., Lim M. H., Zhu N. and Wong K. W. (2006) *Classification of adaptive memetic algorithms: a comparative study.* IEEE Transactions on Systems, Man, and Cybernetics, Part B, vol. 36, no. 1, pp. 141-152.

Özcan E., Bilgin B., and Korkmaz E. E. (2006) *Hill climbers and mutational heuristics in Hyper-heuristics.* In Proceedings of the 9th International Conference on Parallel Problem Solving from Nature, vol. 4193 of Lecture Notes in Computer Science, pp. 202-211.

Özcan E., Bilgin B. and Korkmaz E. E. (2008) *A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis*, vol.12, no.1, pp.3-23.

Özcan E., Bykov Y., Birben M. and Burke E. K. (2009) *Examination timetabling using late acceptance hyper-heuristics.* In Proceedings of Congress on Evolutionary Computation, pp. 997-1004.

Pal N. R., Chakraborty S. and Bagchi A. (1997) *RID3: An ID3-like algorithm for real data.* Information Sciences, vol. 96, pp. 271-290.

Pappa G. L. and Freitas, A. A. (2004) *Towards a genetic programming algorithm for au-tomatically evolving rule induction algorithms.* In Proceedings of ECML/PKDD-2004 Workshop on Advances in Inductive Learning, pp. 93-108.

Pappa G. L. and Freitas A. A. (2006) *Automatically evolving rule induction algorithms.* In Proceedings of 17th ECML, LNCS, vol. 4212, pp. 341-352.

Pappa G. L. (2007) *Automatically Evolving Rule Induction Algorithms with Grammar-based Genetic Programming.* PhD Thesis, Computing Laboratory, University of Kent, Canterbury, UK.

Pattipati K. R. and Alexandridis M. G. (1990) *Application of heuristic search and information theory to sequential fault diagnosis.* IEEE Transactions on Systems, Man and Cybernetics, vol. 20, no. 4, pp. 872-887.

Petrovic S. and Qu R. (2002) *Case-based reasoning as a heuristic selector in a hyperheuristic for course timetabling problems.* In Sixth International Conference on Knowledge-based Intelligent Information and Engineering Systems.

Pfahringer B., Bensusan H. and Giraud-Carrier C. (2000) *Meta-learning by landmarking various learning algorithms.* In Proceedings of the 17th International Conference on Machine Learning, pp. 743 -750.

Pham T. H., Ho T. B. (2007) *A Hyper-heuristic for Descriptive Rule Induction.* International Journal of Data Warehousing and Mining, vol. 3, no. 1, pp. 54-66.

Pillay N. and Banzhaf W. (2007) *A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem.* In Progress in Artificial Intelligence, 13th Portuguese Conference on Artificial Intelligence, Springer, Lecture Notes in Computer Science, vol. 4874, pp. 223-234.

Pillay N. (2008) *An analysis of representations for hyper-heuristics for the uncapacitated examination timetabling problem in a genetic programming system.* In Proceedings of the 2008 Annual Conference of the South African Institute of Computer Scientists and Information Technologists on IT

Research in Developing Countries, Wilderness, South Africa, vol. 338, pp. 188-192.

Pisinger D. and Ropke S. (2007) *A general heuristic for vehicle routing problems.* Computers & Operations Research, vol. 34, no. 8, pp. 2403-2435.

Poli R. (2008) *Some Ideas about No-Free Lunch for Hyper-Heuristics.* Technical Report from University of Essex.

Poli R., Woodward J. R. and Burke E. K. (2007) *A histogram-matching approach to the evolution of bin-packing strategies.* In Proceedings of the Congress on Evolutionary Computation, Singapore, pp. 3500-3507.

Poli R. and Graff M. (2009) *There is a free lunch for hyper-heuristics, genetic programming and computer scientists.* In European Conference on Genetic Programming, pp. 195-207.

Qu R. and Burke E. K. (2005) *Hybrid Variable Neighborhood HyperHeuristics for Exam Timetabling Problems.* In The Sixth Metaheuristics International Conference, Vienna, Austria.

Qu R. and Burke E. K. (2008) *Hybridisations within a graph based hyper-heuristic framework for university timetabling problems.* Journal of the Operational Research Society.

Quinlan J. R. (1979). *Discovering rules by induction from large collections of examples.* In Expert Systems in the Micro Electronic Age, Edinburgh University Press, pp. 168-201.

Quinlan J. R. (1986) *Induction of Decision Trees.* Machine Learning, vol. 1 no. 1, pp. 81-106.

Quinlan J. R. and Rivest R. L. (1989) *Inferring decision trees using the minimum description length principle.* Information and Computation, vol. 80, no. 3, pp. 227-248.

Quinlan J. R. (1987). *Simplifying decision trees.* International Journal of Man-Machine Studies, vol. 27, pp. 221-234.

Quinlan J. R. (1993) *C4.5: Programs for Machine Learning.* Morgan Kaufmann, Los Altos CA, US.

Quinlan J. R. (1998) *Data mining tools See5 and C5.0.* Technical Report, RuleQuest Research.

Raileanu L. E. and Stoffel K. (2004) *Theoretical Comparison between the Gini Index and Information Gain Criteria.* Annals of Mathematics and Artificial Intelligence, vol. 41, pp. 77-93.

Reeves C. (1996) *Hybrid genetic algorithms for bin-packing and related problems.* Annals of Operations Research, vol. 63, pp. 371-396.

Risannen J. (1989) *Stochastic Complexity in Statistical Enquiry.* World Scientific Series in Computer Science, 15.

Ross P., Schulenburg S., Marin-Blázquez J. G. and Hart E. (2002) *Hyper-heuristics: learning to combine simple heuristics in bin-packing problem.* In Proceedings of the 2002 Genetic and Evolutionary Computation Conference, Morgan-Kauffman.

Ross P., Marín-Blázquez J. G., Schulenburg S. and Hart E. (2003) *Learning a procedure that can solve hard bin-packing problems: a new GA-based approach* *to hyper-heuristics.* In Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation, part II.

Ross P., Marín-Blázquez J. G. and Hart E. (2004) *Hyper-heuristics applied to class and exam timetabling problems.* In Proceedings of the 2004 IEEE Congress on Evolutionary Computation, IEEE Press, pp. 1691-1698.

Ross P. (2005) *Hyper-heuristics. In Search Methodologies: Introductory Tutorials in Optimization and Decision Support Methodologies*, Springer, chapter 17, pp. 529-556.

Ross P. and Marín-Blázquez J. G. (2005) *Constructive hyper-heuristics in class timetabling.* In IEEE Congress on Evolutionary Computation, IEEE, pp. 1493-1500.

Rounds E. (1980) *A combined non-parametric approach to feature selection and binary decision tree design.* Pattern Recognition, vol. 12, pp. 313-317.

Schaffer J. D. and Eshelman L. J. (1996) *Combinatorial optimisation by genetic algorithms: The value of the genotype/phenotype distinction.* In Modern Heuristic Search, pp. 85-97.

Schmiedle F., Drechsler N., Große D. and Drechsler R. (2002) *Heuristic learning based on genetic programming.* Genetic Programming and Evolvable Machines, vol. 4, pp. 363-388.

Sethi I. K. and Sarvarayudu G. P. R. (1982) *Hierarchical classifer design using mutual information.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 4, pp. 441-445.

Shing M. T. and Parker G. B. (1993) *Genetic algorithms for the development of real-time multi-heuristic search.* In Fifth International Conference on Genetic Algorithms, pp. 575-572.

Smith D. (1985) *Bin packing with adaptive search.* In Proceedings of the First International Conference on Genetic Algorithms and their Applications, pp. 202-207.

Smyth P. and Goodman R. M. (1991) *Rule induction using information theory.* Knowledge Discovery in Databases, AAAI/MIT press.

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

Stephenson M., O'Reilly U., Martin M. and Amarasinghe S. (2003) *Genetic Programming Applied to Compiler Heuristic Optimization.* In Proceedings of the 6th European Conference on Genetic Programming, Essex, UK.

Storer R. H., Wu S. D. and Vaccari R. (1992) *New search spaces for sequencing problems with application to job shop scheduling.* Management Science, vol. 38, no. 10, pp. 1495-1509.

Storer R. H., Wu S. D. and Vaccari R. (1995) *Problem and heuristic space search strategies for job shop scheduling.* ORSA Journal of Computing, vol. 7, no. 4, pp. 453-467.

Suyama A., Negishi N. and Yamaguchi T. (1998) *CAMLET: A Platform for Automatic Composition of Inductive Learning Systems Using Ontologies.* In Pacific Rim International Conference on Artificial Intelligence, pp. 205-215

Swain P. and Hauska H. (1977) *The decision tree classifer design and potential.* IEEE Transactions on Geoscience and Electronics, vol. 15, pp. 142-147.

Syswerda G. (1991) *Schedule optimization using genetic algorithms.* In Handbook of Genetic Algorithms, New York, pp. 333-349.

Talmon J. L. (1986) *A multiclass nonparametric partitioning algorithm.* Pattern Recognition Letters, vol. 4, pp. 31-38.

Tavares J., Machado P., Cardoso A., Pereira F. B. and Costa E. (2004) *On the evolution of evolutionary algorithms.* In Proceedings of the European Conference on Genetic Programming, Coimbra, Portugal, pp. 389-398.

Tay J. C. and Ho N. B. (2008) *Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems.* Computers and Industrial Engineering, vol. 54, no. 3, pp. 453-470.

Terashima-Marín H., Ross P. and Valenzuela-Rend´on M. (1999) *Evolution of constraint satisfaction strategies in examination timetabling.* In Genetic and Evolutionary Computation Conference, pp. 635–642.

Terashima-Marín H., Morán-Saavedra A. and Ross P. (2005) *Forming hyper-heuristics with GA s when solving 2D-regular cutting stock problems.* In Proceedings of the Congress on Evolutionary Computation, pp. 1104-1110.

Terashima-Marín H., Zárate C. J. F., Ross P. and Valenzuela-Rendón M. (2006) *A GA-based method to produce generalized hyper-heuristics for the 2D-regular cutting stock problem.* In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 591-598.

Terashima-Marín H., Zárate C. J. F., Ross P. and Valenzuela-Rendón M. (2007) *Comparing two models to generate hyper-heuristics for the 2d-regular bin-packing problem.* In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 2182-2189.

Terashima-Marín H., Ortiz-Bayliss J. C., Ross P. and Valenzuela-Rendón M. (2008) *Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems.* In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 571-578.

Thabtah F. and Cowling P. (2008) *Mining the data from a hyperheuristic approach using associative classification.* Expert Systems with Applications: An International Journal, vol. 34, no. 2, pp.1093-1101.

Varshney P. K., Hartmann C. R. P. and De Faria Jr J. M. (1982) *Applications of information theory to sequential fault diagnosis.* IEEE Transactions on Computation, vol. 31, no. 2, pp. 164-170.

Vázquez-Rodríguez J. A., Petrovic S. and Salhi A. (2007a) *A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines.* In Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2007).

Vázquez-Rodríguez J. A., Petrovic S. and Salhi A. (2007b) *An investigation of hyperheuristic search spaces.* In Proceedings of the IEEE Congress on Evolutionary Computation, pp. 3776-3783.

Vella A., Corne D. and Murphy C. (2009) *Hyper-heuristic Decision Tree Induction.* World Congress on Nature & Biologically Inspired Computing, pp. 409-414.

Voudouris C. and Tsang E. (1999) *Guided local search and its application to the traveling salesman problem.* European Journal of Operational Research, vol. 113, no. 2, pp. 469–499.

Wan R., Takigawa I., and Mamitsuka H. (2006) *Applying Gaussian distribution-dependent criteria to decision trees for high-dimensional microarray data.* In Proceeedings of Data Mining in Bioinformatics Workshop at the 32nd International Conference on Very Large Databases, vol. 4316, pp. 40-49.

Wang Q. R. and Suen C. Y. (1984) *Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 6, pp. 406-417.

Zhengou W. and Yan L. (1993) *A new inductive learning algorithm: Separability-Based Inductive learning algorithm.* Acta Automatica Sinica, vol. 5, no. 3, pp. 267-270.

White A. P. and Liu W. Z. (1994) *Technical note: Bias in information-based measures in decision tree induction.* Machine Learning, vol. 15, no. 3, pp. 321-329.

Wilson S. (1995). *Classifier fitness based on accuracy.* Evolutionary Computation, vol. 3, no. 2, pp. 149-175.

Wolpert D. H. and Macready W. G. (1997) *No Free Lunch Theorems for Optimization.* IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 67-82.

Zhou X. J. and Dillon T. S. (1991) *A statistical-heuristic feature selection criterion for decision tree induction.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 13, no. 8, pp. 834-841.

Zhang W. and Dietterich, T. G. (1995) *A reinforcement learning approach to job-shop scheduling.* In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 1114-1120.

Zhang W. and Dietterich, T. G. (1995) *High-performance job-shop scheduling with a time-delay TD(λ) network.* In Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference, pp. 1024-1030.