Automatic Heuristic Generation with Genetic Programming: Evolving a Jack-of-all-Trades or a Master of One

Edmund K. Burke
ekb@cs.nott.ac.uk

Matthew Hyde
mvh@cs.nott.ac.uk

Graham Kendall
gxk@cs.nott.ac.uk

John Woodward
jrw@cs.nott.ac.uk

The University of Nottingham
School of Computer Science and Information Technology
Jubilee Campus
Wollaton Road
Nottingham, UK

ABSTRACT

It is possible to argue that online bin packing heuristics should be evaluated by using metrics based on their performance over the set of all bin packing problems, such as the worst case or average case performance. However, this method of assessing a heuristic would only be relevant to a user who employs the heuristic over a set of problems which is actually representative of the set of all possible bin packing problems. On the other hand, a real world user will often only deal with packing problems that are representative of a particular sub-set. Their piece sizes will all belong to a particular distribution. The contribution of this paper is to show that a Genetic Programming system can automate the process of heuristic generation and produce heuristics that are human-competitive over a range of sets of problems, or which excel on a particular sub-set. We also show that the choice of training instances is vital in the area of automatic heuristic generation, due to the trade-off between the performance and generality of the heuristics generated and their applicability to new problems.

Categories and Subject Descriptors

D.2.2 [**Artificial Intelligence**]: Automatic Programming—*program synthesis*

General Terms

Design, Performance, Reliability

Keywords

Bin-Packing, Genetic Programming, Hyper-Heuristics, Heuristics

1. INTRODUCTION

There are many real world examples where an agent (for example, a factory worker or a robot on a production line) is trained to excel at a specialised task but is not suited to carry out other tasks, even if the tasks appear to be similar. There is often a trade-off between how specialised an agent is at a given task and how well they can adapt to other tasks. An example of this is the education system. This is general at the beginning and specialisation increases the further you progress through the system. This intuitive observation is summarised in the phrase, *jack of all trades and master of none*, which is used to describe an individual who is capable of a range of tasks, but who does not excel at any one in particular.

In this paper, we hypothesise that the same is true when evolving heuristics for the bin packing problem using our Genetic Programming (GP) hyper-heuristic system. The one dimensional bin-packing problem involves a set of integer-size pieces L, which must be packed into bins of a certain capacity C, using the minimum number of bins possible. In other words, the set of integers must be divided into the smallest number of subsets so that the sum of the sizes of the pieces in a subset does not exceed C [18]. In this paper, the 'on-line' bin packing problem is studied. That is, we do not know in advance how many pieces there are or the size of those pieces. Our system must simply pack the pieces into the bins in the order they arrive, and the pieces cannot be moved once they have been placed in a bin [7].

The bin packing problem is known to be NP-Hard [13] so heuristics are commonly used to generate solutions that are of a high enough quality for practical purposes, as a polynomial-time exact algorithm is unlikely to exist for the general case [7]. Examples of online bin packing heuristics can be found in [23, 7, 14]. No known heuristic has both a better worst case performance ratio and average uniform case performance ratio (with items drawn uniformly in the interval [0,1]) than 'best-fit' [15]. This heuristic puts the piece in the fullest bin that has room for it and opens a new bin if the piece does not fit into any existing bin. This heuristic will act as a benchmark to show that the heuristics presented in this paper perform as well as human-designed heuristics.

Evolutionary approaches have been applied to bin packing problems with success [10, 19]. In these applications, the solution is evolved directly by the heuristic. In contrast, this paper will present a hyper-heuristic which evolves bin packing heuristics, and does not evolve solutions directly. Human competitive heuristics are produced automatically, which can then be re-used on new problems without any further time-intensive evolution, which is not true of previous evolutionary approaches in this domain. This paper will attempt to show the reader the benefits and implications of this automatic heuristic generation process, and that a heuristic can be evolved to be a 'jack of all trades', or a master of one.

2. HYPER-HEURISTIC BACKGROUND

Hyper-heuristics can be thought of as heuristics which choose "between a set of low-level heuristics, using some learning mechanism" [27]. Hyper-heuristic research is concerned with search across spaces of heuristics [24]. One of the aims of a hyper-heuristic [24, 1] is to "raise the level of generality at which optimisation systems can operate". They can also be thought of as automatic heuristic generators, where the aim of the hyper-heuristic is to produce an independent heuristic for a problem, that is capable of being re-used.

The No Free Lunch theorem [29, 28] shows that all search algorithms have the same average performance over all problems defined on a given finite search space. However, it is important to recognise that this theorem is *not* saying that it is not possible to build search methodologies which are *more* general than is currently possible. Indeed, research into hyper-heuristics is motivated by the assertion that in many real world problem solving environments, there are users who are interested in *"good-enough soon-enough cheap-enough"* solutions to their optimisation problems [1].

We will briefly discuss some examples of previous hyper-heuristic methods that have appeared in the literature. Two hyper-heuristic methods have been tested on the one dimensional bin-packing problem, a learning classifier system [26] and a genetic algorithm [25]. Simulated annealing is used as a hyper-heuristic in [9] for the shipper rationalisation problem. A case based reasoning hyper-heuristic is used in [6] for both exam timetabling and university course timetabling. Three new hyper-heuristic architectures are presented in [20], which treat mutational and hill climbing low-level heuristics separately. A graph based hyper-heuristic is presented in [5]. In [4], a tabu search hyper-heuristic is presented and evaluated upon a nurse scheduling problem and a university course timetabling problem. A choice function has also been employed as a hyper-heuristic, to rank the low-level heuristics and choose the best one [8], and a distributed choice function hyper-heuristic is presented in [22]. The choice function considers the recent effectiveness of each heuristic and each pair of heuristics, and also considers the time since the heuristic was last called.

These examples represent methodologies that involve supplying the hyper-heuristic with a set of low level heuristics that it can choose between when deciding which heuristic to use at a given decision point. This approach provides a solution to each problem that the hyper-heuristic is applied to. The benefit of this approach is that the strengths of the low level heuristics can be combined and the results are potentially superior to simply using each low level heuristic in isolation. The disadvantage is that it is not always obvious which low level heuristics to give to the hyper-heuristic. In many cases the best heuristic (or set of heuristics) for the problem will not be known.

```
For each problem s in set S
  For each piece p in L
    variable bestSoFar = negativeInfinity
    For each bin i in A
      output = evaluate(bin i)
      If (output > bestSoFar)
        bestSoFar = output
        bestBinIndex = i
      End If
    End For
    Put piece in bestBinIndex bin
  End For
FitnessOfHeuristic += fitness of result of s
End For
```

Figure 1: Pseudo code showing the overall program structure within which a heuristic operates

This paper builds upon our work in [3] where we present a genetic programming system for solving bin-packing heuristics. This is distinct from most of the hyper-heuristics discussed in the rest of this section. There is little other previous work in this area in the literature. The best example is 'CLASS', developed by Fukunaga, which is an automatic generator of local search heuristics for the SAT problem, competitive with human-designed heuristics in terms of runtime and search efficiency [11, 12].

3. THE GP HYPER-HEURISTIC

3.1 Evolving the Choice of Bin

Our system evolves a control program that rates each bin on its suitability for the piece. An individual is assessed by rating the results created when the algorithm in Fig. 1 is run, where S = the set of 20 instances, L = the list of all the pieces in the current instance, and A = the array of bins. The GP parameters are summarised in table 1.

Note that when making the choice of which bin to put the current piece into, each individual in the population is not constrained by whether it is legal to do so. For example, putting every piece in the first bin is permitted. However, this will lead to an illegal solution with a high penalty. For this reason, when the *best-of-run* individual produces a legal solution, it is because the system will have evolved an understanding of the rules, not because of constraints imposed by humans.

3.1.1 Initialisation Parameters.

A population size of 1000 was chosen because this makes the initial population diverse enough, and it allows for reasonable run times. The initial trees are limited to a maximum depth of 4, and the bloat effect is limited by using a very simple method, similar to the 'Tarpeian wrapper' method [21]. Any individual 30 nodes larger than the average of the population is penalised, by setting its fitness to the same high penalty that is given to a heuristic that produces an illegal solution. The 'Grow' method of initialising the trees [17] is used, and we ran the evolution for 50 gener-

Table 1: The GP parameters

Population Size	1000
Generations	50
Crossover Proportion	90%
Reproduction Proportion	10%
Selection Method	Roulette wheel
Initial Maximum Tree Depth	4

ations, a standard parameter used in [16]. These parameters were chosen from a range of possible combinations, after a series of experiments, because they result in good solutions in reasonable time.

The function set is $\{\leq, +, -, \times, \%\}$ where % is the 'protected divide' function [16], and $\leq$ returns 1 if true and -1 if false. There are three terminals, the capacity of the bin, the fullness of the bin, and the size of the piece.

3.1.2 Fitness measure.

The fitness measure is shown in equation 1, where: n = number of bins, $fullness_i$ = sum of all the pieces in bin i, and C = bin capacity

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

This fitness function is taken from [10]. It puts a premium on bins that are filled completely or nearly so. Importantly, the fitness function avoids the problem of plateaus in the search space that occur when the fitness function is simply the number of bins used by the heuristic, which is a problem encountered in [2]. Solutions with lower fitness are better. A fitness of 1000 is assigned to any illegal solution (an arbitrarily high number compared to the range of fitness values for a legal solution). Another option would have been to use a penalty proportional to the degree of violation. For example, if a heuristic filled bins over their capacity, we could assign a penalty proportional to the percentage that the bin is overfilled, or the percentage of bins that are overfilled. We use a fixed penalty for all illegal heuristics because we do not wish to distinguish between heuristics which violate the hard constraints of the problem to different degrees. This is to enhance the simplicity of the algorithm, and because we do not believe it would be beneficial for components of *any* illegal heuristic to be used in the next generation, regardless of the extent to which they violate the hard constraints.

3.1.3 Genetic Operators.

At the end of each generation, 10% of the next generation is created with the reproduction operator (cloning), and 90% is created by crossover. These are standard parameters taken from [16]. In the crossover operator, any node in the tree can be selected with equal probability to be the crossover point. 'Fitness proportional selection' is employed [16] using the normalised fitness, and reselection is permitted.

4. THE PROBLEM CLASSES

We use uniform distributions of piece sizes to define problem classes. The classes are referred to as C_{l-u} where l is the lower limit of the uniform distribution and u is the upper limit. The piece sizes used in our randomly generated instances are shown graphically in Fig. 2. We generated the dataset in this way because we found no dataset in the literature containing such a hierarchy structure, which allows us to test the specialisation of a heuristic over three levels. Each instance has 120 pieces, and the bin capacity is always 150. Heuristics are always evolved by evaluating their performance on a particular set of 20 bin packing instances. Each group of 20 instances is referred to as a 'set' of instances. The defining characteristic of a set is the uniform distribution that its piece sizes are taken from. A distribution will be referred to as a 'class'.

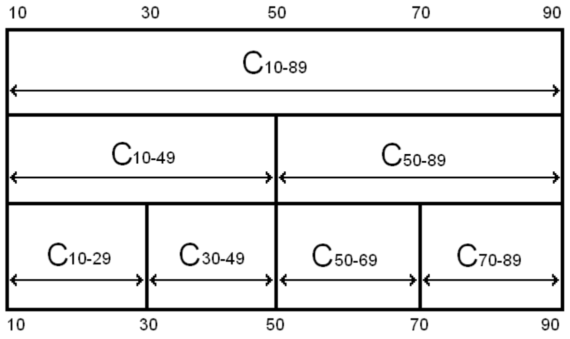

Figure 2: Diagram of the piece size distributions of each class

There are 7 sets involved in training the heuristics, each from a different class, and therefore there are 7 classes that the heuristics can be evolved to be applicable for. The 7 classes represent 3 levels of generality as shown in Fig. 2. We refer to C_{10-49} and C_{50-89} as sub-problems of C_{10-89}. In the same way, C_{10-29} and C_{30-49} are sub-problems of C_{10-49}, and C_{50-69} and C_{70-89} are sub-problems of C_{50-89}. This is because, as can be seen in Fig. 2, all of the values in a sub-problem can appear in its super-problem, but not all of the values that appear in the super-problem can appear in the sub-problem. So the sub-problems are less general as the pieces have a smaller range of values.

In our experiments, we created 7 training sets and 7 validation sets, from the 7 classes. The validation sets have the same format as the training sets, but with new piece size values taken from the same uniform distributions. They are 7 new problem sets taken from the same 7 classes as the training sets. Training sets and validation sets are referred to as T_{l-u} and V_{l-u}, where l is the lower limit of the uniform distribution and u is the upper limit.

To obtain a heuristic, we run the GP on a training set of instances. This is done 50 times for each set. Each set of 50 heuristics produced is known as H_{l-u} where l and u are the lower and upper limits of the uniform distribution of the training set they were evolved on. The result is 350 heuristics in total.

In the training stage, a heuristic will only come into contact with one training set. In the validation stage, the heuristic is separately tested on all 7 validation sets. This is to investigate how re-usable the heuristic is on new problems of the same class and problems of different classes.

5. RESULTS AND DISCUSSION

The results are presented in three sections, each addressing a different aspect of how the evolved heuristics perform

Table 2: The performance of each set of evolved heuristics tested against best-fit

Validation Set	Better than Best-Fit	No Significant Difference	Worse than Best-Fit
H_{10-89}	0	48	2
H_{10-49}	0	50	0
H_{50-89}	0	49	1
H_{10-29}	5	45	0
H_{30-49}	6	44	0
H_{50-69}	0	29	21
H_{70-89}	3	47	0

Table 3: A comparison of the performance of six sets of evolved heuristics with the heuristic sets evolved on their respective super-class and super-super-class. The values represent the probabilities that the performance values of both heuristic sets come from underlying populations with the same mean

Heuristic Set	Super-Class	Super-Super-Class
H_{10-49}	$3.86 * 10^{-4}$	N.A
H_{50-89}	0.35	N.A
H_{10-29}	$9.11 * 10^{-3}$	$6.15 * 10^{-6}$
H_{30-49}	0.29	$3.95 * 10^{-3}$
H_{50-69}	0.09	$1.86 * 10^{-6}$
H_{70-89}	$7.15 * 10^{-10}$	$1.06 * 10^{-6}$

on new instances that they have not been evolved on. In this section we refer to the hierarchy of problem classes shown in Fig. 2. We discuss going up and down levels of generality in this hierarchy and we also refer to the concept of two classes being unrelated if they share no common range of numbers. There are two classes in the middle layer of the hierarchy, the four classes at the bottom of the hierarchy each have a super-class and an unrelated class in the middle layer.

5.1 Quality of the heuristics on new problems of the same class

In this section, we are only concerned with the results of each heuristic on the validation set of the class that it was evolved on, and how they compare to the results of best-fit on that validation set.

We compare each heuristic to best-fit by using a two-tailed paired-difference t-test, using the 20 pairs of results that the heuristic and best fit produce on the 20 instances of the validation set. This test applies here over a standard t-test because the values in each sample of 20 results are not independent. The two results in each pair depend upon the specific piece sizes of the problem instance.

Table 2 is a summary of the performance of the 50 heuristics evolved for each class of problem, and shows the number of heuristics that perform significantly better, significantly worse, and not significantly different, to best-fit. A 0.05 significance level is used in the test, below which we discard the null hypothesis, that the mean of the differences between the performances over the 20 instances is zero.

The majority of heuristics are competitive with best-fit. However, there are 14 evolved heuristics that perform significantly better than best-fit on their validation sets. These are all produced from training on classes at the bottom of the hierarchy, i.e. the narrowest problem distributions. From this we conclude that it is more probable for a heuristic to become specialised in these classes due to the piece size distribution being sufficiently narrow.

The exception appears to be the set H_{50-69}, from which 21 heuristics perform significantly worse than best-fit on V_{50-69}, and none perform better than best-fit. We attribute the anomalous result on V_{50-69} to the good performance of best-fit on this class of piece sizes. It would seem that best-fit performs well when the pieces are all roughly between 1/3 and 1/2 of the bin capacity, and so finding a heuristic that on average outperforms best-fit is more difficult in this class.

5.2 Quality of the heuristics on new problems of different classes

In this section we analyse how heuristic sets perform on the validation set from the class they were evolved on, compared to the performance of heuristic sets evolved on the super-class and super-super-class of that validation set. For example, the super-class of P_{10-29} is P_{10-49}, and its super-super-class is P_{10-89}. The super-class of P_{50-89} is P_{10-89}, and has no super-super-class (indicated by N/A in table 3). The performance of a heuristic on each instance of the validation set is totalled to obtain a total score for the heuristic. Therefore there are 50 total scores for each heuristic set. This set of 50 scores is compared to the 50 total scores of the other set to assess if there is a significant difference in performance between the two.

The results shown in table 3 are the results of a one-tailed t-test between the set of 50 total scores of one heuristic set and the 50 total scores of another. In each comparison, we are testing the hypothesis (H_1) that the values of the sub-class come from a distribution lower (better) than the distribution of the super-class. The null hypothesis (H_0) is that the values of both the sub-class and the super-class come from distributions with the same mean. The values shown represent the probability that H_0 is true. We use a 0.05 significance level in the discussion below.

Table 3 shows that as the classes become more specialised, the heuristics evolved on them can be more specialised too. This hierarchy is observed in all situations except for one anomaly, where set H_{10-89} is significantly better at V_{50-69} than H_{50-69}. In contrast to the other results in table 3, we use a two-tailed t-test for this comparison of H_{50-69} with its super-super-class. This is because the mean of H_{50-69} is higher (worse) than the mean of its super-super-class, and therefore we test whether the distributions are significantly different in either direction, as opposed to testing H_1. The lesser known full epithet is *Jack of all trades, master of none, though ofttimes better than master of one*. The distribution of the piece sizes of V_{50-69} and their ratio with the capacity of the bins present a situation which is obviously one of the times where the jack of all trades is better than the master of one. The heuristics which represent the jack of all trades in this case are able to use their wider 'skills' to perform better on average than heuristics which are specialised in class C_{50-69}.

There is no statistically significant difference between the mean performances of H_{50-89}, H_{50-69} and H_{30-49} on their native classes compared to the performances of heuristics evolved on their super-classes. There seems to be no specific features of these classes that heuristics can learn through

Table 4: Summary of the illegal results when heuristic sets are each tested on three unrelated validation sets, containing piece sizes they did not pack in their evolution. A cross represents at least one illegal result, a circle represents no illegal results

	Validation sets of three classes unrelated to the heuristic set		
Heuristic Set	Mid-level	Two low-level classes	
H_{10-49}	×	×	×
H_{10-29}	×	×	×
H_{30-49}	×	×	×
H_{50-89}	×	×	○
H_{50-69}	○	○	○
H_{70-89}	○	○	○

evolution that they cannot already learn by packing the pieces of their super-class. In our experiments, it appears that piece sizes of around 1/3 of the bin capacity apparently inhibit the specialisation of heuristics on these classes. Conversely, there appears to be more scope for specialisation at the extremes of the range of piece sizes used here. Specifically, H_{10-29} and H_{70-89} are the only sets of heuristics with means that are significantly better than their super-class and their super-super-class at the 98% level of confidence.

5.3 Heuristic Robustness

Often the robustness of a heuristic is important in a commercial scenario. Where there are financial considerations, practitioners may be reluctant to trust in an evolved heuristic to perform well on new problems that it is given. We recognise this issue and hope to present some reassurance. The results of this work show how important it is that the training set used to evolve a heuristic is representative of the future problems that the heuristic is expected to encounter. The results show that if a heuristic is presented with pieces it has not seen before, then there is a chance it will not be able to function as intended and illegal solutions may result. However, the results also show that no heuristic generated automatically by our GP method produced an illegal result when applied to new problems of the same class.

All the evolved heuristic sets were tested on all of the validation sets, as explained in section 4, but table 4 presents only the results where the heuristic set is tested on an unrelated validation set, containing piece sizes that the heuristics did not pack during their evolution. In table 4, a cross represents at least one illegal result on the validation set by the 50 heuristics in the set. A circle represents no illegal results by that heuristic set on the validation set.

H_{50-89} produces some illegal results on V_{10-29} but not on V_{50-69} and V_{70-89}. In the same way, H_{10-49} produces some illegal results on V_{50-69} and V_{70-89} but not on V_{10-29} and V_{30-49}. H_{10-89}, This is summarised in the first and fourth rows of table 4. The most general set of heuristics, produces no illegal results over any of the validation sets. These results show that heuristics evolved on a super-class could be applied to sub-classes without them failing. However, when applied to an unrelated class, there is a chance that the heuristic will not produce a legal solution. This can be explained by the fact that when applied to an unrelated class, the heuristics will be trying to pack pieces that they have not been evolved to pack and there is a chance that a heuristic may try to use relationships of piece sizes, fullness, and capacity that cannot exist in the new class.

To help explain the next point, let set A be the group of classes C_{10-49}, C_{10-29} and C_{30-49}. Also, let set B be the group of classes C_{50-89}, C_{50-69} and C_{70-89}. Table 4 shows that set A seems to be easier for the heuristics trained on set B than set B is for the heuristics trained on set A. This can be seen by comparing the top three rows of table 4 to the bottom three rows. For example, on set A, only 2 heuristics from H_{50-69} result in some illegal solutions, and they only result in illegal solutions on V_{10-49} and V_{10-29}. In contrast to this, H_{10-49}, H_{10-29}, and H_{30-49} all result in some illegal solutions on each of V_{50-89}, V_{50-69} and V_{70-89}.

Our results show that the techniques evolved for the distributions above 1/3 of the bin capacity can mostly be applied to lower piece sizes (piece sizes of less than 50), even if these techniques do not result in equivalent performance in the lower distributions.

The reverse is true for the bin packing techniques developed for lower piece size distributions. The heuristics that embody these techniques are less applicable to problems with higher piece sizes and result in more illegal solutions.

Table 5: Summary of the results obtained by the heuristic shown in equation 2 compared to the results of best-fit

Validation set	Heuristic result	Best-fit result
V_{10-89}	835	834
V_{10-49}	676	503
V_{50-89}	1220	1220
V_{10-29}	2400	326
V_{30-49}	702	704
V_{50-69}	1198	1198
V_{70-89}	1815	1815

5.4 Example Heuristic

Equation 2 shows a simplified representation of a 35 node individual which was evolved on T_{10-90}. Table 5 shows the results that this heuristic obtains on the seven validation sets.

$$\frac{2S+F}{S+F} + \frac{C}{((\frac{F}{C}) \leq (2C-F)) + (C-S-F)} \qquad (2)$$

The table shows that the heuristic uses at most 1 more bin than best fit on each validation set, apart from sets V_{10-29} and V_{10-49}. Indeed, the result for V_{10-29} represents the heuristic putting each piece in a new bin. Given a piece size between 10 and 29, no fullness between 10 and 29 can make the heuristic evaluate to a result greater than it does when the fullness is 0. So the 'rating' that the heuristic gives to the bins with a piece in will always be lower than the rating that it gives to the empty bin. Therefore, no piece will ever be put in a bin which already contains a piece, it will always be placed in an empty bin.

However in all other classes, where the pieces can be larger, there are combinations of piece sizes and fullness that can make the heuristic value greater than when the fullness is zero. So it is possible for pieces to be put in bins that already contain a piece, but still, no piece of less than 30 will ever be put in a bin with a fullness of less than 30. The heuristic's result on V_{10-49} also shows this difficulty that the heuristic has when packing small pieces, however the result is better than for V_{10-29} because some of the pieces are large enough for the fullness to reach a high enough value, so that more pieces can be put in the same bin.

It is worth noting that while the result for V_{10-29} is very poor, none of the solutions are illegal. The safeguard against illegal solutions is the right hand term of the two terms in equation 2, and it is a safeguard that holds in all problem classes. If putting a piece in a bin would overfill the bin by 1 unit, then the right hand term becomes equal to 1, which is low enough for the whole heuristic to value an empty bin more, so the heuristic will never choose a bin that would be overfilled over a bin that the piece can fit in. If the bin would be overfilled by 2 or more units, then the value of the whole heuristic becomes negative, and again the heuristic will rate the open bin more highly than it rates an illegal placement.

Equation 2 is an example of a heuristic strategy that uses ratios of piece sizes, bin fullness and bin capacity that exist in the class it was evolved on, but can not exist in certain other classes.

6. SUMMARY AND CONCLUSIONS

The results of this paper lead to three main conclusions. Firstly, on new instances of a particular class, a heuristic evolved on instances drawn from that class will perform better on average than a heuristic evolved on instances from a different class. Secondly, heuristics can be evolved to be specialists on a particular sub-problem, or general enough to work on all sub-problems. However there is a trade-off between performance and generalisation. Thirdly, heuristics evolved on a super-class will not produce illegal results on problems from one of its sub-classes. However there is a chance that if a heuristic is applied to problems of a different class then illegal solutions may be produced.

These three points show, above all, that care must be taken to ensure that a representative training set is provided when automatically evolving heuristics. The heuristics generated by this type of system will all be specialised to some degree, and just because a heuristic can solve a bin packing problem, does not mean it can solve all bin packing problems.

It is also important to note that the heuristics evolved here have learned how not to violate the hard constraints of the bin packing problem. The heuristics are free to do so, but their evolved strategies guard against this violation.

Research into conventional hyper-heuristics is indicative of how difficult it is to predict which heuristic will work best on a particular problem. The hyper-heuristic is there to automate that decision process. The next step is to develop systems capable of automatically generating the heuristics themselves, as Fukunaga has shown [11, 12], and as we have shown here with a system that can generate heuristics which are competitive with the human-created best-fit heuristic, over a range of different bin packing problems.

The main contribution of this paper is to highlight implications for future systems such as this in other domains. When we are using uniform distributions of piece sizes that fit neatly together in a hierarchy, it may seem obvious to which class a training set of instances belongs to, and therefore one can be fairly confident of which other classes a heuristic evolved on that set will perform well on. However, in messy real-world timetabling and scheduling domains, for example, the constraints and parameters are so varied that it is much more difficult to say that an instance is a member of a certain class. Certain subtle, unforeseen features of a new problem instance may cause a pre-evolved heuristic to produce much worse solutions than expected, even if the new problem is judged to belong to the same class as the training set.

7. FUTURE WORK

We wish to extend this work to offline bin packing, where the heuristics are, in general, more complex than for online algorithms. We intend to develop a grammar-based GP system which will be capable of expressing most of the man-made bin packing heuristics described in the literature. When such a grammar is in place, the system will be able to develop more complex heuristics than is possible with the technique described in this paper.

In contrast to the conventional hyper-heuristic approaches and other meta-heuristics, our GP hyper-heuristic approach has two outputs, good quality solutions, and a heuristic that can be re-used on future problems of the same type without further training. The additional output of a re-usable heuristic means that the computationally expensive evolution process can be employed to produce a heuristic on small problems, and then the resulting heuristic can be applied quickly to larger problem instances. In this way, the solutions to the larger instances could potentially be obtained much more efficiently than if an evolutionary algorithm is employed on the larger instances directly. We intend to investigate further this potential benefit of the heuristic generation process.

8. ACKNOWLEDGMENTS

This work was carried out under ESPRC grant reference EP/C523385/1.

9. REFERENCES

[1] E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Meta-Heuristics*, pages 457–474. Kluwer, 2003.

[2] E. K. Burke, M. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In T. Runarsson, H.-G. Beyer, E. Burke, J. J.Merelo-Guervos, D. Whitley, and X. Yao, editors, *LNCS 4193, Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, pages 860–869, Reykjavik, Iceland, September 2006.

[3] E. k. Burke, M. R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In T. Runarsson, H.-G. Beyer, E. Burke, J. J.Merelo-Guervos, D. Whitley, and X. Yao, editors, *LNCS 4193, Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, pages 860–869, Reykjavik, Iceland, September 2006.

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

[5] E. K. Burke, B. McCollum, A. M. S. Petrovic, and R. Qu. A graph-based hyper heuristic for timetabling

problems. *European Journal of Operational Research*, 176:177–192, 2007.

[6] E. K. Burke, S. Petrovic, and R. Qu. Case-based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132, 2006.

[7] E. G. Coffman Jr, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Combinatorial analysis. In D. Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*. Kluwer, 1998.

[8] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. K. Burke and W. Erben, editors, *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT 2000)*, pages 176–190, Konstanz, Germany, August 2000.

[9] K. Dowsland, E. Soubeiga, and E. K. Burke. A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research*, 179(3):759–774, 2007.

[10] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings of the IEEE 1992 Int. Conference on Robotics and Automation*, Nice, France, May 1992.

[11] A. Fukunaga. Automated discovery of composite sat variable-selection heuristics. In *Eighteenth national conference on Artificial intelligence*, pages 641–648, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[12] A. S. Fukunaga. Evolving local search heuristics for SAT using genetic programming. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. Tyrrell, editors, *Proceedings of the Genetic and Evolutionary Computation Conference 2004 (GECCO '04)*, volume 3103, pages 483–494, Seattle, WA, USA, 2004. Springer-Verlag.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Fransisco, 1979.

[14] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. Worst-case performance bounds for simple one-dimensional packaging algorithms. *SIAM Journal on Computing*, 3(4):299–325, December 1974.

[15] Kenyon. Best-fit bin-packing with random order. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, 1996.

[16] J. R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. The MIT Press, Boston, Massachusetts, 1992.

[17] J. R. Koza and R. Poli. Genetic programming. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 127–164. Springer, Boston, 2005.

[18] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, Chichester, 1990.

[19] M. O'Neill, R. Cleary, and N. Nikolov. Solving knapsack problems with attribute grammars. In *Proceedings of the Third Grammatical Evolution Workshop (GEWS'04)*, Seattle, WA, USA., 2004.

[20] E. Ozcan, B. Bilgin, and E. E. Korkmaz. Hill climbers and mutational heuristics in hyperheuristics. In T. Runarsson, H.-G. Beyer, E. Burke, J. J.Merelo-Guervos, D. Whitley, and X. Yao, editors, *LNCS 4193, Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, pages 202–211, Reykjavik, Iceland, September 2006.

[21] R. Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Genetic Programming, Proceedings of the 6th European Conference, EuroGP 2003*, pages 211–223, Essex, April 2003. Springer-Verlag.

[22] P. Rattadilok, A. Gaw, and R. Kwan. Distributed choice function hyper-heuristics for timetabling and scheduling. In E. Burke and M.Trick, editors, *Practice and Theory of Automated Timetabling V, Springer Lecture notes in Computer Science*, volume 3616, pages 51–67, 2005.

[23] W. T. Rhee and M. Talagrand. On line bin packing with items of random size. *Math. Oper. Res.*, 18:438–445, 1993.

[24] P. Ross. Hyper-heuristics. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 529–556. Springer, Boston, 2005.

[25] P. Ross, J. G. Marin-Blazquez, S. Schulenburg, and E. Hart. Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyperheurstics. In *Proceedings of the Genetic and Evolutionary Computation Conference 2003 (GECCO '03)*, pages 1295–1306, Chicago, Illinois, 2003.

[26] P. Ross, S. Schulenburg, J. G. Marin-Blazquez, and E. Hart. Hyper heuristics: Learning to combine simple heuristics in bin packing problems. In *Proceedings of the Genetic and Evolutionary Computation Conference 2002 (GECCO '02)*, 2002.

[27] E. Soubeiga. *Development and Application of Hyperheuristics to Personnel Scheduling*. PhD thesis, Univesity of Nottingham, School of Computer Science, 2003.

[28] D. Whitley and J. P. Watson. Complexity theory and the no free lunch theorem. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Kluwer, Boston, 2005.

[29] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 4:67–82, 1997.