Generating Single and Multiple Cooperative Heuristics for the One Dimensional Bin Packing Problem Using a Single Node Genetic Programming Island Model

Kevin Sim
Institute for Informatics and Digital Innovation
Edinburgh Napier University
Edinburgh, Scotland, UK
k.sim@napier.ac.uk

Emma Hart
Institute for Informatics and Digital Innovation
Edinburgh Napier University
Edinburgh, Scotland, UK
e.hart@napier.ac.uk

ABSTRACT

Novel deterministic heuristics are generated using Single Node Genetic Programming for application to the One Dimensional Bin Packing Problem. First a single deterministic heuristic was evolved that minimised the total number of bins used when applied to a set of 685 training instances. Following this, a set of heuristics were evolved using a form of cooperative co-evolution that collectively minimise the number of bins used across the same set of problems. Results on an unseen test set comprising a further 685 problem instances show that the single evolved heuristic outperforms existing deterministic heuristics described in the literature. The collection of heuristics evolved by cooperative co-evolution outperforms any of the single heuristics, including the newly generated ones.

Categories and Subject Descriptors
Computing methodologies [Machine learning]: Machine learning algorithms

Keywords
Hyper-Heuristics; One Dimensional Bin Packing Algorithms; Single Node Genetic Programming

1. INTRODUCTION

This paper introduces a system for generating both discrete and cooperative heuristics for application to the 1D Bin Packing Problem (BPP). A compact form of Genetic Programming (GP) named Single Node Genetic Programming (SNGP) [16, 15] is used as a generative hyper-heuristic (HH) to create novel deterministic heuristics. The motivation for the research lies with the premise that selective HH can exploit the abilities of different heuristics by selecting the one that is best suited for each problem instance. Also generative HH can be used to generate single heuristics which exhibit superior performance than similar human designed heuristics and can be used to produce collections of heuristics that collectively maximise the potential of selective HH. The papers contribution is two fold. Single heuristics capable of outperforming a range of well researched deterministic heuristics are generated by combining features extracted from those heuristics. Second an island model[19] is adapted to use multiple SNGP implementations to generate diverse sets of heuristics which collectively outperform any of the single heuristics when used in isolation. Both approaches are trained and evaluated using equal divisions of a large set of 1370 benchmark problem instances sourced from the literature[11, 22]. The heuristics generated outperform 6 well researched human-designed deterministic heuristics in terms of both the number of optimum solutions found in the test set, and the total number of bins required to solve all instances.

2. ONE DIMENSIONAL BIN PACKING

The one dimensional bin packing problem (BPP) is a well researched NP-hard problem [13] which has been tackled using exact procedures[22], deterministic heuristics[13], biologically inspired metaheuristics [11, 18] and HH [20, 1, 21, 5, 24]. The objective is to find the optimal number of bins, \( OPT(bins) \), of fixed capacity \( c \) required to accommodate a set of \( n \) items, \( J = \{ \omega_1 \ldots \omega_n \} \) with weights \( \omega_j : j \in \{1 \ldots n\} \) falling in the range \( 1 \leq \omega_j \leq c \) whilst enforcing the constraint that the sum of weights in any bin does not exceed the bin capacity \( c \). For any instance \( OPT(bins) \) must lie between the lower and upper bounds shown in Equation 1 with the upper bound occurring when all items are greater than half the bin capacity and the lower bound achieved when the total free space summed across all bins is less than the capacity of one bin.

\[
\left\lfloor \frac{\sum_{j=1}^{n} \omega_j}{c} \right\rfloor \leq OPT(bins) \leq n \tag{1}
\]

Metaheuristics, such as Genetic Algorithms (GA) [11] and Ant Colony Optimisation [18], have been successfully used to solve the BPP. However they are often complex hybrid procedures incorporating both stochastic global search and highly optimised local search mechanisms tailored to the benchmark problem instances they are designed to solve. Metaheuristics have been criticised as being costly in terms of the time and the level of expertise required to develop and adapt them for real world applications.
HH, introduced in the following section, aim to address many of the issues surrounding the complexities associated with applying metaheuristics by finding “good enough - soon enough - cheap enough” [2] methods for solving problem instances of widely varying characteristics.

3. HYPER-HEURISTICS

The term hyper-heuristics (HH) first appeared in relation to combinatorial optimisation problems in [6] although the term was first coined in [9] to describe an amalgamation of artificial intelligence techniques in the domain of automated theorem proving. However, the concept can be traced back to the 1960’s when Fisher & Thompson [12] used machine learning techniques to select combinations of simple heuristics to produce solutions to local job-shop scheduling problems. Originally described as “heuristics to select heuristics” [2] the field has evolved to encompass techniques including “heuristics to generate heuristics” using GP to create new heuristics from constituent component parts [4, 5]. All HH, no matter the approach, have the commonality that they search over a landscape defined by a set of heuristics, or their component parts, for a procedure to solve a problem rather than searching directly over the space defined by the problem itself. A more concise review can be found in [2, 3].

HH have been applied to the BPP previously not only in the 1D form investigated here but also in its 2D and 3D varieties. Ross et al.[21], developed a HH that introduced the notion of describing the state of a problem instance according to the percentage of items that fall into 4 pre-defined “natural” categories relating to item size, given as a ratio of the bin capacity. A Michigan style Learning Classifier System (LCS) was used to evolve a set of rules mapping problem states to suitable heuristics. Each iteration the chosen heuristic packs a single bin with the potential of a filler process being invoked that attempts to fill a partially filled bin further. The remaining items are then reclassified using the new problem state resulting in the application of a sequence of heuristics for solving each instance.

In [24] the authors built on the work presented in [21] by training a $k$-nearest neighbour classification algorithm to predict which from a set of four deterministic heuristics would perform best on a large set of benchmark problem instances. Rather than utilising the pre-determined “natural” characteristics used in [21] a genetic algorithm was used to evolve an undetermined quantity of characteristics. The characteristics evolved were expressed as variable sized ranges of the bin capacity and were shown to improve the accuracy when predicting the best heuristic for an instance.

Both HH described above can be characterised as heuristics to select heuristics using the classification schema set out in [4]. In contrast the approaches taken in [1, 5] fall into the class of heuristics to generate heuristics where GP techniques were employed as a means of automating the design of on-line heuristics for the BPP. In both of these studies a small number of 1D BPP benchmark instances were used to evaluate the performance of the heuristics generated. The research presented here is evaluated against a far larger set of problem instances which in conjunction with the fact that the heuristics generated are off-line heuristics means that no direct comparison can be made. In order to evaluate the approach taken here the results obtained are compared to a range of off-line heuristics sourced from the literature. It should be noted that these include a hand crafted heuristic (ADJD) created for a previous study [24] in order to address the poor performance of the other heuristics used on problem instances where the average item size is small when measured as a ratio of the bin capacity.

Heuristics are generated using a compact form of GP, described in Section 5, to evolve both single and innovatively a set of cooperative off-line heuristics which collectively cover more of the problem search space.

In [1, 5] the heuristics evolved directly decide which bin to pack an item into given the state of any partially packed bins and the characteristics of the next item presented. In contrast the heuristics evolved here indirectly pack items, one bin at a time, by means of a side effect of the heuristic.

The result of executing a heuristic is not used explicitly to decide whether to pack an item; rather, the value returned determines whether the packing process currently running should terminate or continue. As a consequence of executing a heuristic certain terminal nodes may be invoked which may cause one more more items to be packed into the current bin; an outcome described in the literature as a “side effect”.

4. BENCHMARKS

Six deterministic heuristics and five sets of benchmark problem instances were sourced from the literature and used to evaluate the heuristics evolved here. Three of the data sets totalling 1210 instances were introduced in [22] and have optimal solutions that vary from the lower bound given by Equation 1. All optimal solutions are known and have been solved since their introduction [23]. A further 2 data sets comprising 160 problem instances were taken from [11]. All but one have optimal solutions at the lower bound [14]. For the remainder of this paper the term “optimal” refers to the solutions described within these publications. The complete set of 1370 benchmark instances were split into equal sized training and test sets with the test set comprising of every 2nd instance. This ensured an even distribution of instances from each of the data sets and from each of the subsets that were generated using discrete parameter combinations. The reader is directed to the original publication for a description of how these problem instances were generated.

Six deterministic heuristics, described below, are used for comparison. All but one are off-line algorithms. The exception is the Sum of Squares (SS) algorithm [7, 8] which is designed as an on-line algorithm but used here as a deterministic off-line heuristic. All heuristics described here, are presented with each problem instance’s items pre-sorted in descending order of size.

- **First Fit Descending** (FFD) packs each item into the first bin that will accommodate it. If no bin is available a new bin is opened. All bins remain open for the duration of the procedure.

- **Djang and Finch** [10] (DJD) packs items into a bin until it is at least one third full. The set of up to three items which best fills the remaining space is then found with preference given to sets with the lowest cardinality. The bin is then closed and the procedure repeats using a new bin.

- **DJD more Tuples** (DJT) [21]. DJT works as for DJD but considers sets of up to five items after the bin is filled more than one third full.
- Adaptive DJD (ADJD) [24] packs items into a bin until the free space is less than or equal to three times the average size of the remaining items. It then operates like DJD looking for the set of up to three items of lowest cardinality that best fills the remaining space.

- Best Fit Descending (BFD) works as for FFD but packs each item into the bin with the least free space which will accommodate it.

- Sum of Squares (SS) [7, 8] is an on-line bin packing heuristic which puts items into bins such as to minimise the number of bins with equal free space. It is included here as a deterministic off-line heuristic with items presented in descending order of size.

It is interesting to note that SS solves 79 of the 80 instances in Falkenauer’s Triplet set when presented with the items in the order that they are published. When the order of items was shuffled randomly, SS failed to find any optimal solution for any of the 80 problem instances. When SS was applied to the 685 problem instances in the test set with the items presented in the order that they are published it finds 421 optimal solutions using 61231 bins. When items are pre-sorted in descending order of size it finds 383 optimal solutions using a total of 61369 bins. When items were shuffled randomly 100 times. SS finds on average 97 optimal solutions using an average of 63282 bins.

5. SNGP

Single Node Genetic Programming was introduced by Jackson in [16] and applied to 3 problems amenable to being solved using dynamic programming, namely 6 multiplexer, even-parity and symbolic regression. SNGP differs from the conventional GP model introduced by Koza [17] in a number of key respects.

- Each individual node may be the starting point for evaluation, not only the top most node.

- Nodes may have any number of parent nodes (including none and duplicates) allowing for network structures other than trees to be formed.

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

Figure 1 shows two partial SNGP structures with any nodes not connected to the current top node omitted for clarity. The standard tree structure on the left show how the DJD heuristic could be represented using the nodes outlined in Table 1. The right side of the diagram highlights a key difference between SNGP and conventional GP; that individual nodes are permitted to have multiple parent nodes. In keeping with the terminology used in the literature each node is considered as an individual in a population. Each node can be the starting point for evaluation making each a unique heuristic. As described later, each SNGP structure contains exactly 1 of each of the available terminal nodes and a predefined number of randomly selected terminal nodes. These may be disconnected during the mutation process. When evaluating a node only it and those nodes connected recursively as child nodes need to be considered. Evaluation of the population may either be carried out by averaging the fitness of all nodes or by considering the node with the best fitness in isolation. The second elitist measure is adopted here.

One of the advantages of SNGP is in its efficiency as all nodes do not need to be re-evaluated after each mutation. However this only holds for problems where the numerical output of an evolved program is used to explicitly make some decision. Jackson carried out a further investigation of SNGP in [15] where its effectiveness on problems where the solution is obtained as a side effect was explored. For this class of problem executing the program does not yield a result which determines an appropriate action but causes a side effect such as producing a transition between adjacent positions in a maze. Three such problems were tackled using SNGP; the Santa Fe artificial ant problem, a maze navigation task and third problem where the objective was to generate a program capable of parsing arithmetic and logical expressions. SNGP was shown to be an effective approach for tackling this class of problem. The results presented showed significant improvement to those obtained using conventional GP. SNGP operates as follows.

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

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

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

4. A single mutation operator is used which selects a function node at random and reassign one of its edges to point at a node chosen randomly from the set of all nodes with a lower identification number.

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

Steps 4 and 5 are repeated until the stopping criteria is met which for both experiments described in Section 7 was after 500 generations. In the two HHI studies, described in Section 3, the value output from the evolved program was used to directly decide where to pack the next item. The system presented here differs in that the process of evaluating a terminal node may cause items to be packed into the current bin.
In order to ensure that all heuristics terminate, even when no items are packed, a wrapper, described by Algorithm 1, is used to encompass the node being evaluated. The wrapper is responsible for determining when to open a new bin based on both the value returned by the heuristic and the state of the solution currently being constructed. The nodes selected for use in this study, described in Table 1, were identified by decomposing the heuristics outlined in Section 4.

Table 1: Nodes Used

<table>
<thead>
<tr>
<th>Function Nodes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Protected divide. Returns -1 if the denominator is 0 otherwise the result of dividing the first operand by the second is returned</td>
</tr>
<tr>
<td>&gt;</td>
<td>Returns 1 if the first operand is greater than the second or -1 otherwise</td>
</tr>
<tr>
<td>IGTZ</td>
<td>Is Greater Than Zero: If the first operand evaluates as greater than zero then the result of evaluating the second operand is returned. Otherwise the result of evaluating the third operand is returned</td>
</tr>
<tr>
<td>&lt;</td>
<td>Returns 1 if the first operand is less than the second or -1 otherwise</td>
</tr>
<tr>
<td>X</td>
<td>Returns the product of two operands</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Terminal Nodes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>Packs the single largest item into the current bin returning 1 if successful or -1 otherwise</td>
</tr>
<tr>
<td>B2</td>
<td>Packs the largest combination of exactly 2 items into the current bin returning 1 if successful or -1 otherwise</td>
</tr>
<tr>
<td>B2A</td>
<td>Packs the largest combination of up to 2 items into the current bin giving preference to sets of lower cardinality. Returns 1 if successful or -1 otherwise</td>
</tr>
<tr>
<td>B3A</td>
<td>As for B2A but considers sets of up to 3 items</td>
</tr>
<tr>
<td>B5A</td>
<td>As for B2A but considers sets of up to 5 items</td>
</tr>
<tr>
<td>C</td>
<td>Returns the bin capacity</td>
</tr>
<tr>
<td>FS</td>
<td>Returns the free space in the current bin</td>
</tr>
<tr>
<td>INT</td>
<td>Returns a constant integer value randomly initialised from $\in [-1, 1, 2, 3, 4, 5]$</td>
</tr>
<tr>
<td>W1</td>
<td>Packs the smallest item into the current bin returning 1 if successful or -1 otherwise</td>
</tr>
</tbody>
</table>

Algorithm 1 SNGP Node Wrapper

Require: $I \in \{i_1, i_2, ..., i_n\}$ \{The set of items to be packed\}

Require: $B = \emptyset$ \{The set of bins which is initially empty\}

repeat
  add a new bin $b$ to $B$
repeat
  $I' = I$
  $result = evaluate(Node)$ \{This may cause items from $I$ to be packed into the current bin $b$\}
until $result < 0$ or $I = \emptyset$ or $I = I'$
if $I = I'$ and $I \neq \emptyset$ then
  pack each remaining item in a new bin
end if
until $I = \emptyset$

SNGP was chosen as a methodology after trials using conventional GP were unsuccessful due to the network structures suffering from bloat. SNGP eliminates this undesirable trait due to its use of a single mutation operator. With no crossover employed there is no mechanism to allow the structures generated to grow beyond their initial size. SNGP also allows for more complex programs (using the same number of nodes) to emerge than would be possible using GP which restricts the topology of networks to tree structures.

For a suitable procedure for solving a problem instance rather than directly searching for a solution. The research presented incorporates an island model with multiple instances of SNGP to generate sets of heuristics which collectively cover this search space better than any single heuristic. The system is explained in the following section.

6. ISLAND MODEL

6.1 Description

The model used here is adapted from [19] where the authors used a novel approach for evolving “interacting co-adapted subcomponents”. The authors distinguish their model from other approaches, such as Learning Classifier systems which the authors describe as competitive rather than cooperative. The model is evaluated on a simple bit string pattern matching task before being applied to the more complex task of evolving weights for a cascading neural network. The bit string pattern matching task involves finding a match set of $N$ binary strings which match as well as possible a much larger target set of $K$ bit strings. The target set is generated at random using either 2, 4 or 8 disparate masks. The objective is to generate a set of either 2, 4 or 8 matching strings. For example if 2 masks are used, 11111111 and 11111111 (where # represents a random binary value), then the objective of is to generate two strings which between them match as many of the bits in this set as possible.

The model allows the number of islands to emerge as a property of the system making the technique amenable in situations where the number of cooperating subcomponents is not known a priori. A matching string is evaluated by averaging the number of bits in the same position with the same value over the complete target set. Islands are removed if their contribution is deemed negligible and are added when the fitness of the system stagnates.

The system described in the following section evolves a set of complimentary heuristics which cover different portions of the heuristic search space and collectively outperform any of the individual constituent heuristics.

6.2 Implementation

Figure 2 illustrates the utility of combining heuristics. Non-overlapping areas describe those instances for which a heuristic uses fewer bins than any other. Heuristic H2 gives no contribution and could be eliminated as it is fully enclosed by H1. The island model described was adapted to use multiple instances of SNGP rather than GAs. Each node in an island’s network structure is evaluated by measuring its ability to cooperate with the best nodes taken from each of the other islands. The process of co-evolving heuristics is described by Algorithm 2 and conceptualised by Figure 3. Note that only partial SNGP structures are depicted due to space restrictions. The fitness value attributed to a heuristic (node) is designed to reflect its ability to cooperate with the best nodes from each of the other islands.
Algorithm 2 Island Model Pseudo-Code

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

7. EXPERIMENTS AND RESULTS

Two sets of experiments were conducted using the set of training instances and then evaluated on the unseen test instances outlined in Section 4. Both experiments were conducted using the island model outlined previously with the first experiment restricted to using a single island.
The objective of the first experiment was to generate a single heuristic which minimised the cumulative number of bins required when applied to all 685 problems in the test set. The second experiment used the island model described previously to co-evolve a set of cooperative heuristics which when applied greedily to the test set were able to collectively outperform any of the individual constituent heuristics.

Both experiments were executed 30 times and each terminated after 500 iterations. The results presented here only show the first 250-260 generations as no improvement was observed after this point. In all cases the SNGP structures were initialised as described in Section 5 using 12 randomly selected function nodes. The software was implemented in Java and executed on a high performance cluster comprising of 18 servers each equipped with dual, quad-core cpu’s with 16Gb ram running Fedora 12.

The results of the first experiments are summarised in Table 2 which shows for comparison the number of problem instances that were solved optimally by each of the six deterministic heuristics described in Section 4. The number of extra bins more than the optimal of 60257 required by each heuristic when applied to the same set of instances is also given. The table also gives the results obtained by treating each terminal node as an individual heuristic. This highlights the benefits of using SNGP to evolve heuristics which are composed of combinations of nodes used in the terminal set. It is interesting to note that by encompassing the nodes in a wrapper that continues to execute the heuristic until no more items are packed even the simple terminal nodes when used in isolation can outperform the human designed heuristics.

Table 2: Comparison between 6 deterministic heuristics, the terminal nodes that cause a side effect and the best generated heuristic (HGEN) on the test set of problem instances.

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Optimal Solutions</th>
<th>Extra Bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFD</td>
<td>393</td>
<td>1088</td>
</tr>
<tr>
<td>DJD</td>
<td>356</td>
<td>1216</td>
</tr>
<tr>
<td>DJT</td>
<td>430</td>
<td>451</td>
</tr>
<tr>
<td>ADJD</td>
<td>336</td>
<td>679</td>
</tr>
<tr>
<td>BFD</td>
<td>394</td>
<td>1087</td>
</tr>
<tr>
<td>SS</td>
<td>383</td>
<td>1112</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Terminal Node</th>
<th>Optimal Solutions</th>
<th>Extra Bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>393</td>
<td>1088</td>
</tr>
<tr>
<td>B2</td>
<td>308</td>
<td>3250</td>
</tr>
<tr>
<td>B2A</td>
<td>432</td>
<td>944</td>
</tr>
<tr>
<td>B3A</td>
<td>303</td>
<td>764</td>
</tr>
<tr>
<td>B5A</td>
<td>332</td>
<td>692</td>
</tr>
<tr>
<td>W1</td>
<td>31</td>
<td>16761</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generated Heuristic</th>
<th>Optimal Solutions</th>
<th>Extra Bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>HGEN</td>
<td>518</td>
<td>257</td>
</tr>
</tbody>
</table>

Figures 4 and 5 show statistical results for single heuristic generation taken over 30 runs. The box plots illustrate how 25 of the 30 runs evolve in less than 250 generations to give the same results using both the number of instances solved and the total number of bins required as metrics. For those 5 runs where the results deviated from the best the deviation was minimal. Using a Mann-Whitney rank sum test to compare the best human designed heuristic (DJT) to the best evolved heuristic using the number of bins required as a metric offers little insight into the obvious improvement in the results due to the fact that both heuristics generate solutions which require an equal number of bins for many of the problem instances. However if Falkenauer’s fitness function [11] is used as a metric then the one-tailed and two-tailed P values obtained show the results to be highly significant measuring at $5.48 \times 10^{-5}$ and $10.96 \times 10^{-5}$ respectively. Two of the best heuristics generated are shown in Figure 6.

The results of the second experiment, designed to generate a set of cooperative heuristics which when applied greedily to the training problems collectively minimise the number of bins required, are shown in Table 3. The number of heuristics evolved is an emergent property of the system and is not predefined. All heuristics contribute towards the combined improvement. The number of optimal solutions found and the number of bins utilised for many of the functions which require an equal number of bins for many of the training instances as a metric.
Table 3: Comparison between 6 man made heuristics and 6 generated heuristics on 685 test problems

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Optimal Solutions</th>
<th>Extra Bins</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFD</td>
<td>393</td>
<td>1088</td>
</tr>
<tr>
<td>DJD</td>
<td>356</td>
<td>1216</td>
</tr>
<tr>
<td>DJT</td>
<td>430</td>
<td>451</td>
</tr>
<tr>
<td>ADJD</td>
<td>336</td>
<td>679</td>
</tr>
<tr>
<td>BFD</td>
<td>394</td>
<td>1087</td>
</tr>
<tr>
<td>SS</td>
<td>383</td>
<td>1112</td>
</tr>
<tr>
<td>Combined</td>
<td>544</td>
<td>186</td>
</tr>
</tbody>
</table>

| H1        | 332               | 692        |
| H2        | 476               | 820        |
| H3        | 465               | 363        |
| H4        | 420               | 500        |
| H5        | 267               | 850        |
| H6        | 471               | 409        |
| Combined  | 559               | 159        |

The table also shows the total number of problems solved and the number of extra bins than the optimal required by each collection of heuristics if applied in a greedy manner. It is clear that the evolved heuristics outperform the human designed ones in terms of both metrics used. In order to provide a further comparison, the best heuristics obtained from each experiment were used to solve a much larger set of 15830 problem instances (available from [http://www.soc.napier.ac.uk/~cs378/bpp/](http://www.soc.napier.ac.uk/~cs378/bpp/)). The best single evolved heuristic used 7.8% fewer extra bins than were required by ADJD which was the best human designed heuristic on these problems. ADJD used 18541 extra bins that the known optimal of 1,362,542. The set of six cooperative heuristics collectively required 21% fewer extra bins (15070) than the optimal number than were needed when greedily applying the best of the man made heuristics.

Figures 7 and 8 show the performance of the island model during training. The darkest line at the top of Figure 7 and the bottom of Figure 8 show the results obtained if the heuristics are applied greedily to each instance. The other plots on each graph show the results obtained on the training set by each of the individual constituent heuristics evolved by the system.

The approach is novel both in its use of SNGP to find new heuristics, and in its use of cooperative co-evolution to find sets of heuristics that interact cooperatively to collectively minimise the number of bins used across the same set of problem instances.

8. CONCLUSIONS AND FUTURE WORK

Single heuristics were generated that outperformed a selection of well researched deterministic heuristics when used to solve a large set of 685 problem instances frequently used as benchmark instances in the literature. Furthermore an island model was added which used a form of cooperative co-evolution to generate a set of novel heuristics that interact cooperatively to collectively minimise the number of bins used across the same set of problem instances.

The approach is novel both in its use of SNGP to find new heuristics, and in its use of cooperative co-evolution to find sets of heuristics. The results attained highlight the utility of using HH to combine simple deterministic heuristics in order to exploit their combined strength. It is intended to expand the approach taken here by incorporating a heuristic selection strategy. This would create a HH which falls into both HH categories; “heuristics to select heuristics” and “heuristics to generate heuristics”.
9. REFERENCES


