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. Secondly 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.

$$\lceil (\sum_{j=1}^{n} \omega_j) \div c \rceil \leq OPT(bins) \leq n \qquad (1)$$

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 2^{nd} 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.

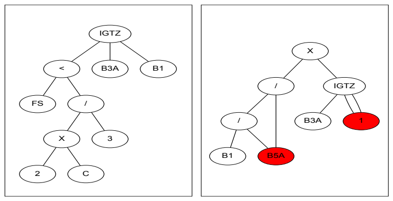

Figure 1: Two examples of partial SNGP structures with any unused nodes hidden.

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

2. A number, n, of function nodes are selected at random from the set of all function nodes $F \in \{f_1, \dots, f_s\}$ and given an identification number ranging from $r+1$, $\dots r+n$. 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 HH 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

Function Nodes

/	Protected divide. Returns -1 if the denominator is 0 otherwise the result of dividing the first operand by the second is returned
$>$	Returns 1 if the first operand is greater than the second or -1 otherwise
IGTZ	Is Greater Than Zero: If the first operand evaluates as greater than zero then the result of evaluating the second operand is returned. Otherwise the result of evaluating the third operand is returned
$<$	Returns 1 if the first operand is less than the second or -1 otherwise
X	Returns the product of two operands

Terminal Nodes

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

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.

HH search 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 coadapted subcomponents". The authors distinguish their model from other approaches, such as Learning Classifier systems which the authors describe as *competitive* rather than *cooperative*. The model is evaluated on a simple bit string pattern matching task 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, ####1111 and 1111#### (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 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.

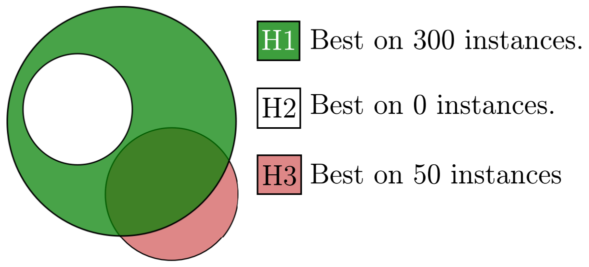

Figure 2: The Venn diagram conceptualises how a set of heuristics collectively improve upon their individual abilities to solve a set of problem instances.

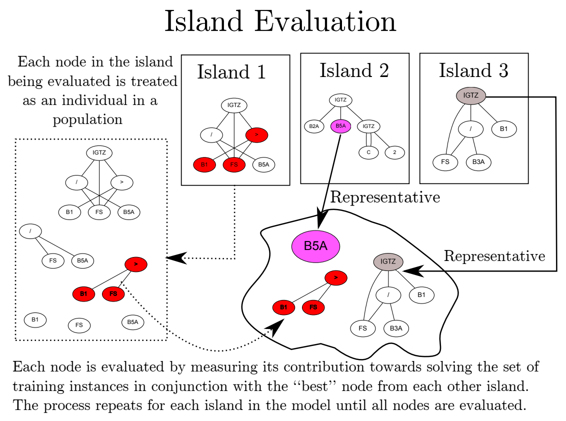
Island Evaluation
Each node in the island being evaluated is treated as an individual in a population
Island 1
Island 2
Island 3
IGTZ
/
>
B1
FS
B5A
B2A
C
2
B3A
Representative
Representative
Each node is evaluated by measuring its contribution towards solving the set of training instances in conjunction with the "best" node from each other island. The process repeats for each island in the model until all nodes are evaluated.

Figure 3: Measuring an Islands Contribution to the Ecosystem.

Fitness is calculated using Equations 2, 3 and 4 and is simply the sum of the number of bins fewer required by the heuristic in comparison to the best result achieved by any of the other islands best heuristics. Only problem instances where the heuristic being evaluated uses less bins than any of the other islands best heuristics are used for the fitness metric's calculation.

$$fitness_{ij} = \sum_{k=1}^{p} \Delta bins_{ijk} \tag{2}$$

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

$$\Delta bins_{ijk} = \begin{cases} best_p - bins_{ijk} & \text{:if } bins_{ijk} < best_p \\ 0 & \text{:otherwise} \end{cases} \tag{3}$$

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

$$best_p = \min\left(best_{i_{bins_p}}\right) \;\; \forall i \in \{1, \ldots, n\} : i \neq x \tag{4}$$

where x is the id of the island being evaluated, $best_{i_{bins_p}}$ is the number of bins required to pack problem p using the node with the best fitness from island i and n is the number of islands in the ecosystem. Each island is evaluated in turn. The diagram shows how each of the six nodes from island 1 (along with their successor nodes) that constitute the 6 *heuristics* are decomposed. Each node from the island being evaluated is placed into a set of nodes containing one representative from each of the other islands. The node selected as a representative from each of the other islands is that which was awarded the highest fitness score during the previous iteration. All nodes in an island have their fitness value recalculated after the island is mutated as follows.

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

This process then repeats with the other islands.

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** $Island \in Islands$ **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** $Island \in Islands$ **do**
    mutate(island)
    evaluate all nodes in mutated island
    set all nodes fitnesses in mutated island
    **if** evaluateBins() $\geq$ 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 heuristics. 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.

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

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 5.48×10^{-5} and 10.96×10^{-5} respectively. Two of the best heuristics generated are shown in Figure 6.

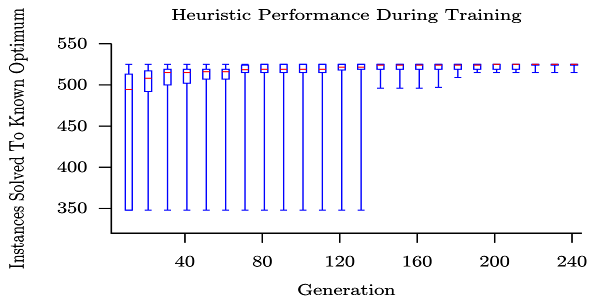

Figure 4: Single heuristic performance over 30 runs using the number of training problem instances solved using the known optimal number of bins as a metric

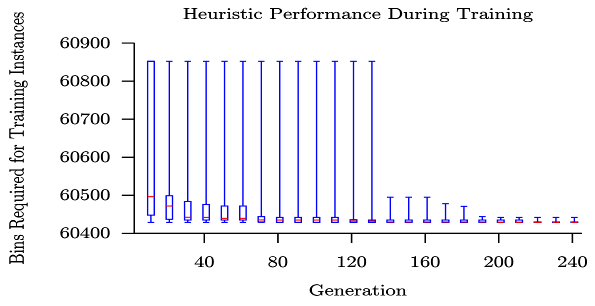

Figure 5: Single heuristic performance over 30 runs using the total number of bins required to pack all training instances as a metric

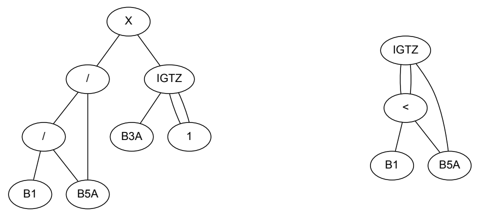

Figure 6: Two of the single heuristics evolved.

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 by each new heuristic are shown. These results are contrasted with the 6 heuristics described in Section 4

Table 3: Comparison between 6 man made heuristics and 6 generated heuristics on 685 test problems

Heuristic	Optimal Solutions	Extra Bins
FFD	393	1088
DJD	356	1216
DJT	430	451
ADJD	336	679
BFD	394	1087
SS	383	1112
Combined	**544**	**186**

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 set of 15830 problem instances (available from `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.

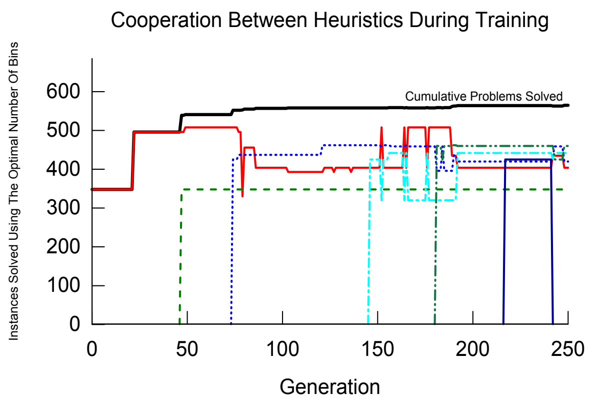

Figure 7: Number of problems solved individually and cumulatively from 685 problem instances used during training

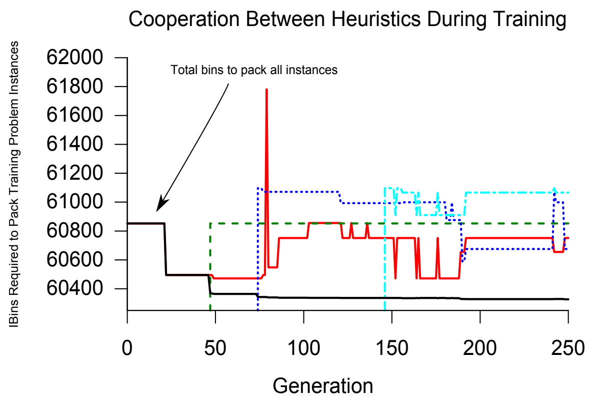

Figure 8: Number of bins used individually and collectively during training.

Figure 9 depicts one of the best sets of 6 heuristics evolved during this experiment which is the set used to obtain the results shown here. Although the results shown are for one individual run, as was the case with the first experiment nearly all of the 30 runs conducted converged to produce identical results which are omitted to increase clarity (27 out of the 30 runs conducted produced sets of heuristics which gave the same results when evaluated using both metrics).

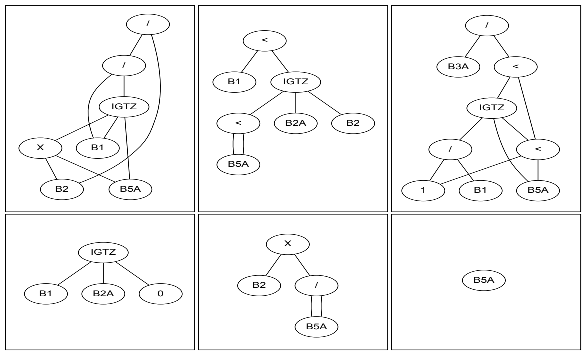

Figure 9: An evolved set of cooperative heuristics with unused nodes omitted for clarity.

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

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

[2] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, chapter 16, pages 457–474. Kluwer, 2003.

[3] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *School of Computer Science and Information Technology, University of Nottingham, Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747.*, 2010.

[4] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A classification of hyper-heuristic approaches. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 449–468. Springer US, 2010.

[5] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. Automating the packing heuristic design process with genetic programming. *Evol. Comput.*, 20(1):63–89, Mar. 2012.

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

[7] J. Csirik, D. S. Johnson, C. Kenyon, J. B. Orlin, P. W. Shor, and R. R. Weber. On the sum-of-squares algorithm for bin packing. *J. ACM*, 53(1):1–65, Jan. 2006.

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

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

[10] P. A. Djang and P. R. Finch. Solving one dimensional bin packing problems, 1998.

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

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

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

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

[15] D. Jackson. A new, node-focused model for genetic programming. In A. Moraglio, S. Silva, K. Krawiec, P. Machado, and C. Cotta, editors, *Genetic Programming*, volume 7244 of *Lecture Notes in Computer Science*, pages 49–60. Springer Berlin Heidelberg, 2012.

[16] D. Jackson. Single node genetic programming on problems with side effects. In C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 327–336. Springer Berlin Heidelberg, 2012.

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

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

[19] M. A. Potter and K. A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evol. Comput.*, 8:1–29, 2000.

[20] P. Ross, J. Marín-Blázquez, S. Schulenburg, and E. Hart. Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics. In E. Cantú-Paz, J. Foster, K. Deb, L. Davis, R. Roy, U.-M. Oï£¡Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation GECCO 2003*, volume 2724 of *Lecture Notes in Computer Science*, pages 215–215. Springer Berlin / Heidelberg, 2003.

[21] P. Ross, S. Schulenburg, J. G. Marín-Blázquez, and E. Hart. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 942–948, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[22] A. Scholl, R. Klein, and C. Jürgens. Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Comput. Oper. Res.*, 24(7):627–645, 1997.

[23] P. Schwerin and G. Wäscher. The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp. *International Transactions in Operational Research*, 4(5-6):377–389, 1997.

[24] K. Sim, E. Hart, and B. Paechter. A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 348–357. Springer Berlin Heidelberg, 2012.