Providing a Memory Mechanism to Enhance the Evolutionary Design of Heuristics

Edmund K. Burke, *Member, IEEE*, Matthew R. Hyde, *Member, IEEE*, and Graham Kendall, *Member, IEEE*

***Abstract*— Genetic programming approaches have previously been employed in the literature to evolve heuristics for various combinatorial optimisation problems. This paper presents a hyper-heuristic genetic programming methodology to evolve more sophisticated one dimensional bin packing heuristics than have been evolved previously. The heuristics have access to a memory, which allows them to make decisions with some knowledge of their potential future impact. In contrast to previously evolved heuristics for this problem, we show that these heuristics evolve to draw upon this memory in order to facilitate better planning, and improved packings. This fundamental difference enables an evolved heuristic to represent a dynamic packing strategy rather than a fixed packing strategy. A heuristic can change its behaviour depending on the characteristics of the pieces it has seen before, because it has evolved to draw upon its experience.**

I. Introduction

Previous work in evolutionary heuristic design for one dimensional bin packing has shown that evolved heuristics perform well on instances similar to those in their training set [1]. In this paper, we investigate whether genetic programming is a suitable technique to evolve heuristics which can draw upon their experience of the pieces they have seen so far, so as to make more informed decisions. The contribution of this paper is to show how to evolve more sophisticated heuristics than have previously been evolved for this problem. These heuristics can draw upon their 'experience' in order to pack more effectively, while previously evolved heuristics cannot use this information. The remainder of the introduction is arranged as follows. Section I-A presents the bin packing problem, and section I-B explains the best-fit heuristic with which we compare our evolved heuristics. Section I-C outlines the contribution of this paper, and section I-D describes the context of this work in relation to other hyper-heuristic work.

A. One Dimensional Bin Packing

The one dimensional bin packing problem consists of a list of pieces which must be assigned to bins with a fixed capacity. The objective is to minimise the number of bins necessary to accommodate all of the pieces [2]. Lower bounds for this problem are given in [3], one of which (L_2) is used in this paper to analyse solution quality. In this paper, the 'online' bin packing problem is studied. That is, we do not know in advance how many pieces there are or the size of those pieces. A heuristic must 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 [4].

Edmund Burke, Matthew Hyde and Graham Kendall are with the Department of Computer Science, University of Nottingham, Nottingham, U.K. (email: [ekb,mvh,gxk]@cs.nott.ac.uk).

The bin packing problem is known to be NP-Hard [5] 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 [4]. Examples of existing bin packing heuristics can be found in [4], [6], [7], and are summarised in [1].

B. The Best-Fit Heuristic

The best-fit heuristic for one dimensional online bin packing places the current piece into the fullest bin that has room for it. As such, it never opens a new bin unless there is no open bin that can accommodate the current piece being packed. Best-fit is used as a comparison in this paper because it represents a successful human-designed general strategy. 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 [8]. It is useful as a benchmark in this paper because it shows the magnitude of the improvement we can achieve with an evolved heuristic which can adapt to the instance it is solving.

C. Evolving Online Bin Packing Heuristics

Previous work has shown that online bin packing heuristics can be evolved which perform better than the best-fit heuristic on similar problems [1]. For example, a heuristic can be evolved on a training set of instances with piece sizes that are uniformly distributed between 10 and 90. We have shown that the evolved heuristic maintains its performance on new problems with the same piece size distribution. Results in [9] show that the evolved heuristics also maintain their performance on instances with a much larger number of pieces than the training set.

In previous work, an evolved online heuristic received one piece at a time, and decided into which bin to pack it. The evolved heuristics represented successful packing strategies that took advantage of the distribution of piece sizes that they saw during their training [1]. They also seemed to 'plan ahead' by often opening a new bin when there was still space for the piece in at least one of the existing bins.

When designing a bespoke heuristic for a given piece size distribution, there is still a difference between the amount of information that those evolved heuristics used, and the amount of information that a human designed heuristic would use. For example, for a problem with piece sizes that are uniformly distributed between 10 and 90, a human would perhaps try to avoid filling a bin so there are 9 units of

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

space left. This would obviously be a waste because a piece would not appear with a size of 9, and the gap would never be filled. Explicit knowledge about the minimum piece size was not available to the heuristics evolved in previous work. As another example, a human performing the packing would gain, from experience, some sense of the probability of a piece of a certain size appearing soon, and so could leave gaps of a given size open in the hope that they could fill these gaps exactly. This is another form of information which was unavailable to the heuristics evolved in previous work.

Best-fit is a fixed strategy, designed as a simple general algorithm which performs reasonably well on any problem instance. Most bin packing problems in the real world would not consist of uniform random pieces, but would contain some structure. For example, an organisation which must cut lengths of metal pipe from fixed sized stock would find that they cut certain lengths more often. This structure can be used to a heuristic designer's advantage, as it allows a heuristic to be 'tailored' to the characteristics of the problem. The structure of the problem can also be learned by remembering the pieces that are seen during the packing, and one could design a heuristic which can perform differently depending on the characteristics of the pieces that it has seen so far.

This design process can be automated. The contribution of this paper is to show how this information can be supplied as building blocks to a genetic programming heuristic design system, and to show the extent to which this information allows more successful heuristics to evolve. This fundamental difference allows a computer designed heuristic to represent a dynamic strategy rather than a fixed strategy. The evolved heuristic can change its behaviour depending on the pieces it has seen, and may perform differently under the same circumstances because it has evolved to draw upon its experience.

D. Hyper-Heuristics

Hyper-heuristics have been defined as heuristics which search a space of heuristics, as opposed to a space of solutions [10], [11], [12]. The genetic programming system we present in this paper can be defined as a hyper-heuristic because it searches the space of heuristics which can be constructed from the functions and terminals. One goal of hyper-heuristic research is to automate the design of heuristic methods. There are at least two classes of hyper-heuristic [13]. The first, and most common in the literature, is the class which intelligently chooses between predefined heuristics. The second class of hyper-heuristic automatically designs new heuristics, more information on this class can be found in [14]. This paper belongs to the second class of hyper-heuristic. Other examples of hyper-heuristics from this class include Fukunaga, who evolves local search heuristics for the SAT problem [15] which are composites of heuristics in previous generations; and Bader-el-Din and Poli who evolve more parsimonious SAT local search heuristics which are faster to execute [16]. Heuristic dispatching rules have also been evolved for the job shop problem [17].

Algorithm 1 How a heuristic produces a solution to each instance and obtains a fitness

```
for all problem instance i in set I do
  initialise solution s as one empty bin
  for all piece p in i do
    variable bestScore = negativeInfinity
    for all bin b in s do
      if p fits into b then
        output = evaluateHeuristic(p, b)
        if output > bestScore then
          bestBin = b
          bestScore = output
        end if
      end if
    end for
    Put piece p in bin bestBin
  end for
  FitnessOfHeuristic += getFitness(s)
end for
```

The work in this paper extends previous work on how to evolve one dimensional bin packing heuristics [1], [9], [18]. Previous work has also shown that human competitive heuristics can be evolved for two dimensional strip packing problems [19], and heuristics designed by evolution for the three dimensional packing problem are compared to a human designed heuristic in [20]. A genetic algorithm is used to evolve hyper-heuristics for the two dimensional packing problem in [21], [22], [23], [24]. The evolved individual contains criteria to decide which packing heuristic to apply next, based on the properties of the pieces left to pack. The genetic algorithm evolves a mapping from these properties to an appropriate heuristic. The work follows studies which evolve similar hyper-heuristics for the one dimensional bin packing problem in [25], [26].

II. Methodology

We employ genetic programming to evolve the decision making criteria of an online bin packing heuristic. An online heuristic takes the pieces of an instance one at a time, and decides where to pack each piece before considering the next piece. The individuals in our genetic programming population are functions represented as tree structures. Each function represents a heuristic strategy for bin packing. To decide where to pack a given piece, the function is evaluated at each available bin in the partial solution, returning a numerical score for each, and the piece is placed into the bin with the highest score. This process is then repeated for every subsequent piece. This process is shown in algorithm 1, and is also explained in previous work [1], [9].

A. The Functions and Terminals

The heuristics evolved in previous work were limited, in the sense that they could not use any information about the pieces that have already been packed. In this paper we investigate if the inclusion of memory can produce heuristics

TABLE I

THE FUNCTIONS AND TERMINALS. ONLY THE FUNCTIONS AND THE STANDARD TERMINALS ARE INCLUDED WHEN EVOLVING A HEURISTIC WITHOUT MEMORY. ALL TERMINALS ARE INCLUDED WHEN EVOLVING A HEURISTIC WITH MEMORY

Type	Symbol	Inputs	Description
Functions	$+$	2	Add two inputs
	$-$	2	Subtract two inputs
	$*$	2	Multiply two inputs
	$\%$	2	Protected divide
Standard Terminals	S	0	Piece size
	E	0	Emptiness of bin (capacity - fullness)
	L	0	Space left in the bin (E-S)
Memory Functions and Terminals	MIN	0	Minimum piece size in memory
	MAX	0	Maximum piece size in memory
	AVE	0	Average piece size in memory
	FE, FL	0	Percent of pieces in memory that would fit into the space E or L
	FXE, FXL	0	Percent of pieces in memory that would *almost exactly* fit into the space E or L (3 or less units wasted)
	FI	1	Percent of pieces in memory that would fit into a space which is the size of the input to this function

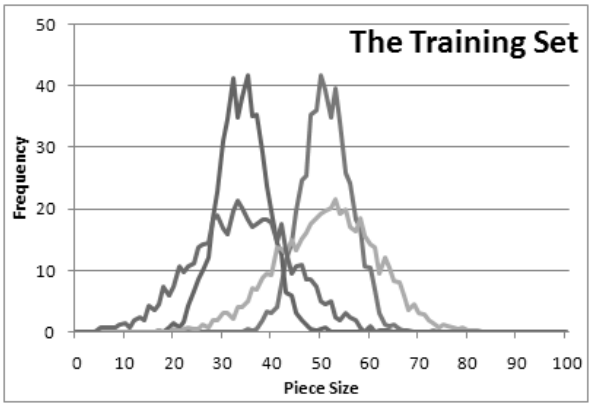

Fig. 1. The average numbers of pieces in the 20 instances of the training set. The training set consists of 4 classes of instance (see table II), and 5 instances are generated from each class. Each training set instance contains 500 pieces. Each graph line represents the average of 5 instances. These are the actual piece size distributions of the instances, and not the probability density functions.

TABLE II

A SUMMARY OF THE INSTANCES IN THE TRAINING SET. THERE ARE 20 INSTANCES IN TOTAL, 5 INSTANCES FROM EACH OF 4 PIECE SIZE DISTRIBUTIONS. THE MEAN AND STANDARD DEVIATION, SHOWN IN THIS TABLE, DEFINE THE GAUSSIAN PIECE SIZE DISTRIBUTION FOR EACH CLASS. THE TABLE ALSO SHOWS THAT THE BIN CAPACITY AND THE NUMBER OF PIECES REMAINS CONSTANT

	Distribution		Bin	No. of
	Mean	S.D.	Cap	Pieces
Class 1 (5 instances)	50	5	100	500
Class 2 (5 instances)	33	5	100	500
Class 3 (5 instances)	50	10	100	500
Class 4 (5 instances)	33	10	100	500

which are more successful and more general. The memory is implemented through the use of a list of the last 100 pieces that have been seen, and a number of genetic programming terminals to allow the heuristics to access certain information about that list.

The genetic programming functions and terminals are shown in table I. As we are investigating whether the inclusion of memory is beneficial, we perform a comparison between heuristics evolved with the memory terminals available, and heuristics with no access to the memory. The 'memory functions and terminals' in table I are omitted from the terminal set when evolving heuristics with no access to the memory.

Recall that a heuristic in the population is a mathematical function, which is evaluated at each bin to produce a score for each bin. The genetic programming terminals represent features of the bin and piece combination that is being considered as the heuristic is evaluated. The functions available to the evolution process are the four arithmetic operators. The standard terminals are S, the size of the current piece; E, the bin emptiness, calculated as the sum of the pieces in the bin (fullness) subtracted from the bin capacity; and L, the space that would be left if the piece were to be put in the bin.

The values of the memory terminals are calculated from the state of the 100 piece memory. The MIN, MAX, and AVE are simple calculations using the past 100 pieces. The other five memory terminals potentially provide the heuristic with the probability of a piece occuring next which will fill a given gap. FE and FXE return the proportion of pieces that the heuristic has seen that would fit into the existing gap. FL and FXL return the proportion of pieces that would fit into the gap left over if the piece was to be placed in the bin. While FE and FL return the proportion of pieces that would fit, FXE and FXL return the proportion of pieces that *almost exactly* fit, defined as leaving a gap of three units or less. In addition to these terminals, we also include a memory function, FI, which performs the same calculation as FE and FL, but does so for an arbitrary size. While FE and FL return the proportion of pieces less than E or L, FI returns the proportion of pieces less than the value of the input to this function.

These memory functions and terminals contain potentially useful information on what may happen in the future if the piece is placed in the bin, given the distribution of pieces that have been seen. For example, if pieces of size 40 occur frequently, and pieces of size 15 do not, we would hope the evolved heuristics would learn that it is not prudent to fill a bin with an emptiness of 40 with a piece of size 25. This would leave a gap of 15, which is unlikely to be filled. Heuristics which use this type of information are potentially much more intelligent, can adapt to different situations, and have a better chance of filling the majority of bins with low waste.

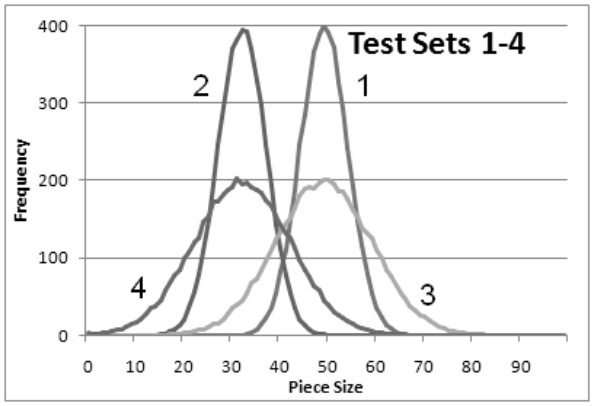

Fig. 2. The average numbers of pieces in the 20 instances of each of the first 4 test sets. Each graph line represents the average of 20 instances, and each instance contains 5000 pieces. These are the actual piece size distributions of the instances, and not the probability density functions.

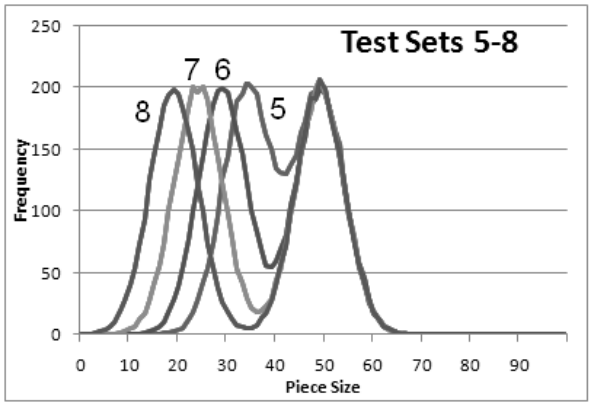

Fig. 3. The average numbers of pieces in the 20 instances of each of the test sets 5 to 8. Each graph line represents the average of 20 instances, and each instance contains 5000 pieces. These are the actual piece size distributions of the instances, and not the probability density functions.

B. Problem Instances

The heuristics are evolved using their performance on 20 training instances, consisting of five instances from each of four different problem classes. The problem classes are shown in figure 1, and described in table II. All training instances contain 500 pieces, and all instances in this study have a constant bin capacity of 100. In this paper, we wish to test if the heuristics can evolve to use information about the distribution of pieces that they have seen so far in the packing, so the piece size distributions must be sufficiently varied to allow, and show, such learning.

There are 12 test sets of instances, each set contains 20 instances, and all test instances contain 5000 pieces. The piece size distributions of all of the test sets can be found in table III, and the pseudocode to generate them is shown in algorithm 2. The first 4 test sets are generated from the same 4 distributions as the training set, because we wish to investigate how well the evolved heuristics generalise to new problem instances similar to those they are evolved on. Test sets 1-4 are shown graphically in figure 2.

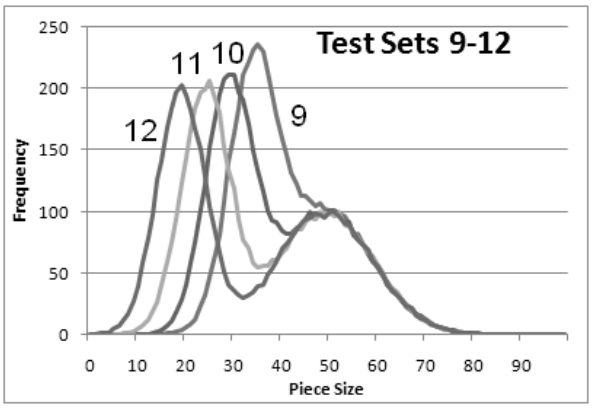

Fig. 4. The average numbers of pieces in the 20 instances of each of the test sets 9 to 12. Each graph line represents the average of 20 instances, and each instance contains 5000 pieces. These are the actual piece size distributions of the instances, and not the probability density functions.

Algorithm 2 Process of generating a test set instance for test sets 1 to 12. Distributions 1 and 2 can be found in table III

```
Create empty piece list L
for i = 1 to 5000 do
  if test set number ≤ 4 then
    L.append(random integer from distribution1)
  else
    if random(0,1) < 0.5 then
      L.append(random integer from distribution1)
    else
      L.append(random integer from distribution2)
    end if
  end if
end for
```

TABLE III

A SUMMARY OF THE INSTANCES IN THE TEST SETS. EACH TEST SET CONTAINS 20 INSTANCES. THE PIECES OF TEST SETS 1-4 ARE RANDOMLY DRAWN FROM THE SAME FOUR GAUSSIAN DISTRIBUTIONS AS THE TRAINING SET. THE PIECES OF TEST SETS 5-12 ARE EACH RANDOMLY DRAWN FROM ONE OF TWO GAUSSIAN DISTRIBUTIONS USING A 0.5 PROBABILITY. THE NUMBER OF PIECES IN EACH INSTANCE IS ALWAYS 5000, MEANING THEY ARE LARGER THAN THE TRAINING SET

	Distribution1		Distribution2		Bin	No. of
	Mean	S.D.	Mean	S.D.	Cap	Pieces
Test Set 1	50	5	N/A	N/A	100	5000
Test Set 2	33	5	N/A	N/A	100	5000
Test Set 3	50	10	N/A	N/A	100	5000
Test Set 4	33	10	N/A	N/A	100	5000
Test Set 5	50	5	35	5	100	5000
Test Set 6	50	5	30	5	100	5000
Test Set 7	50	5	25	5	100	5000
Test Set 8	50	5	20	5	100	5000
Test Set 9	50	10	35	5	100	5000
Test Set 10	50	10	30	5	100	5000
Test Set 11	50	10	25	5	100	5000
Test Set 12	50	10	20	5	100	5000

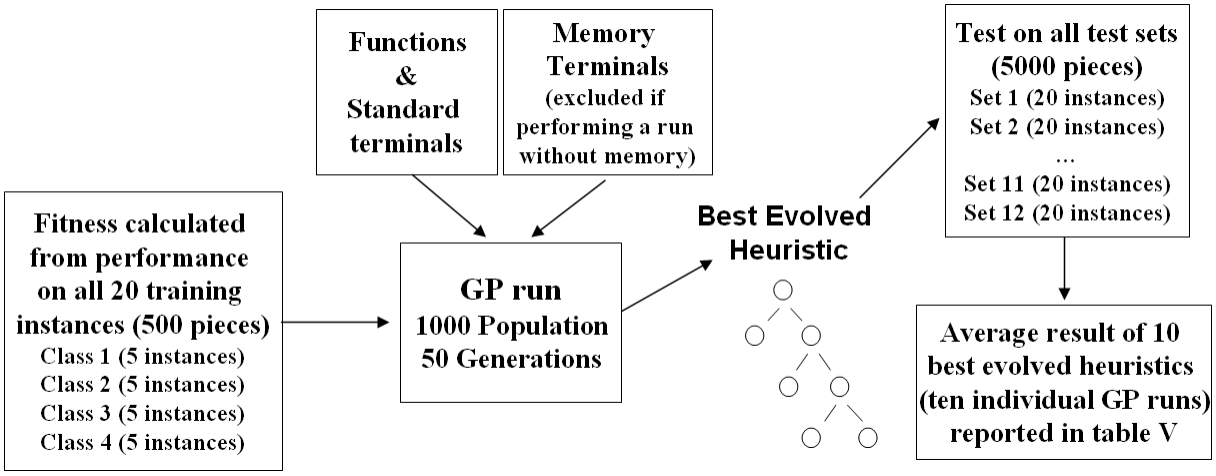

Fig. 5. The process of evolving and testing a heuristic. This occurs once for each evolved heuristic. Ten runs are performed with the memory terminals, and ten runs without the memory terminals. The average results of the two sets of ten are reported in table V.

If the heuristics are expected to evolve to use their memory, and use the probability of pieces of a certain size appearing next, then the evolved heuristics should be able to pack more complex distributions more effectively than heuristics without that information. For this reason, test sets 5-12 consist of bimodal piece size distributions. We wish to investigate whether the lack of pieces in the middle of the distribution can be recognised by the evolved heuristics, even though they have not seen such instances before. The distribution of piece sizes in test sets 5-8 are shown in figure 3, and test sets 9-12 are shown in figure 4.

These instances are used because the maximum and minimum piece size is not enough to capture their distribution. A heuristic would ideally use more than the maximum and minimum piece size to pack these instances well, and so they represent a good test of whether the evolved heuristics are making use of the memory available to them.

C. Genetic Programming System

The genetic programming system is extended from the publicly available ECJ (Evolutionary Computation in Java) package, which can be found at http://cs.gmu.edu/∼eclab/projects/ecj/. The genetic programming parameters are generally default values in that package, which are mostly based on the work by Koza [27]. The parameters are summarised in table IV. For our experiments these parameters result in good results in reasonable run times. We do not claim that the parameters are optimal, and further research can focus on the effects of changing these parameters.

The fitness of a heuristic is calculated from its performance on the 20 instances of the training set. Equation 1 shows how the fitness is calculated. It uses the L_2 lower bound on the instance, which is explained in detail in [3]. It is tighter than the continuous lower bound, as it takes into account the fact that two pieces larger than half the bin capacity cannot be put into the same bin. I is the set of instances in the training set, and n_i is the number of bins used by the heuristic for instance i. A lower fitness is better because the difference between the solutions and their lower bounds will be less.

TABLE IV
THE GENETIC PROGRAMMING PARAMETERS

Population Size	1000
Generations	50
Crossover Proportion	85%
Reproduction Proportion	5%
Mutation Proportion	10%
Selection Method	Tournament, size 7
Initialisation Method	Ramped Half-and-Half
Initial Maximum Tree Depth	2-6

$$Fitness = \sum_{i \in I} \left(\frac{n_i}{L_2(i)} \right) \quad (1)$$

The process of evolving and testing a heuristic is summarised in figure 5. Ten heuristics are evolved without the memory terminals, and ten are evolved with the memory terminals. Therefore, 20 genetic programming runs are performed. Once a heuristic is evolved on the training set, it is then tested on all 12 test sets. The average of each set of ten evolved heuristics is reported in table V.

III. RESULTS

The results are displayed in table V, and graphically in figure 6. Lower values are better, as the results represent the number of bins used by the heuristic. The first aspect of the results to note is that the evolved heuristics are better than best-fit in all but two cases. This corroborates previous work in [1], [9], which shows that evolved heuristics can perform better than best-fit. The heuristics have also been evolved on large instances, of 500 pieces. Previous work has also shown that evolved heuristics maintain their performance on much larger instances when the training set instances contain 500 pieces [9]. The test sets here contain 5000 pieces, and so these results further reinforce the results in [9].

TABLE V

THE AVERAGE RESULTS OF THE EVOLVED HEURISTICS ON THE 12 TEST SETS. THE FIGURES SHOWN ARE THE AVERAGE NUMBER OF BINS USED, OVER THE 20 INSTANCES OF EACH TEST SET. THE RESULTS OF BEST-FIT ARE COMPARED TO THE AVERAGE RESULTS OF THE TEN HEURISTICS EVOLVED WITH NO MEMORY TERMINALS, AND THE AVERAGE OF THE TEN HEURISTICS EVOLVED WITH ACCESS TO THE MEMORY TERMINALS

	Test Set	L2 Lower Bound	Best-Fit	Evolved Without Memory	Evolved With Memory	Significant Difference With Memory
Similar to the Training Set	1	2522.8	2552.6	2552.5	2550.8	Yes ($p<0.001$)
	2	1650.1	1705.0	1681.9	1673.6	Yes ($p<0.001$)
	3	2518.7	2557.9	2558.1	2554.5	Yes ($p<0.001$)
	4	1651.0	1704.2	1683.8	1673.9	Yes ($p<0.001$)
Multimodal Type 1 (figure 3)	5	2124.2	2329.2	2282.3	2258.5	Yes ($p<0.001$)
	6	2000.0	2167.5	2126.9	2086.7	Yes ($p<0.001$)
	7	1876.7	1951.9	1925.6	1910.8	Yes ($p<0.001$)
	8	1755.3	1805.9	1856.3	1781.9	Yes ($p<0.001$)
Multimodal Type 2 (figure 4)	9	2125.6	2306.2	2283.4	2273.6	Yes ($p<0.001$)
	10	2003.0	2148.5	2106.2	2078.2	Yes ($p<0.001$)
	11	1876.9	1973.5	1937.5	1918.1	Yes ($p<0.001$)
	12	1749.0	1803.6	1833.8	1782.5	Yes ($p<0.001$)

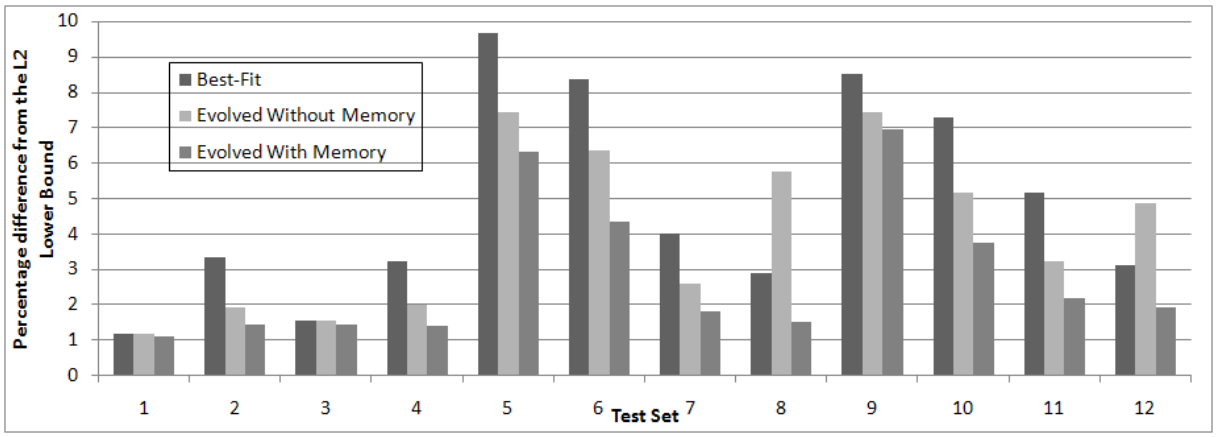

Fig. 6. The results of the best-fit heuristic, compared to the average results of the ten heuristics evolved with no memory terminals, and the average of the ten heuristics evolved with access to the memory terminals. The results are presented as the percentage difference of the result to the L2 lower bound. The results show that the methodology presented in this paper is successful in allowing the heuristics to evolve to take advantage of the information in the memory

We turn our attention now to the contribution of this paper, to show that this genetic programming method can successfully evolve heuristics which make use of their experience (via memory) in order to produce more efficient packings. One can see clearly in figure 6 that the average performance of the heuristics with memory terminals is better in every test set than the heuristics without memory. The heights of the graph bars are always lower, and therefore the results are always closer to the lower bound. From these results, we conclude that the distribution of pieces seen so far can be used by the evolved heuristics to plan more effectively, and that this specific genetic programming system allows this ability to evolve.

We performed a paired difference t-test for each of the 12 test sets, to compare the performance of the 10 heuristics evolved with memory and the 10 heuristics evolved without memory. For each set there are 20 instances, and on each instance we have two average results, each obtained from 10 heuristics. One result is for the heuristics with memory, and the other is from the heuristics without memory. These average results are not independent, they depend on the instance because each instance has a different lower bound. Therefore, for each test set, we must compare the averages of the two sets of heuristics as 20 pairs of results. On every test set, the average results of the heuristics evolved with memory are significantly better than the average results of the heuristics without access to memory teminals.

It is informative to look specifically at the test sets 8 and 12, as best-fit performs better than the evolved heuristics without memory, but not better than the heuristics with

memory. These instances are the 'most' multimodal, as the two Gaussian distributions are furthest apart (see figure 3 and 4), and there are less pieces with sizes in the middle of the distribution. It appears that for these instances, the ability to learn from experience can negate a deficiency shown by the heuristics without memory. The addition of memory appears to allow more robust heuristics to evolve, and this is a key advantage if automatically designed heuristics were to be employed in dynamic environments. A human would naturally adapt a packing strategy to a changing piece size distribution, and this paper has shown how this ability can be evolved.

Consider the results from figure 6 on test sets 5 through 8, and also 9 through 12, as the piece size distribution becomes more bimodal. From these results it appears as though the heuristics perform progressively better, as the bars of the graph become lower. In fact, one would expect the opposite, that the performance of the evolved heuristics would deteriorate as the piece size distribution becomes more bimodal. We would expect this because of previous results in [1] where heuristics are shown to perform better on instances most similar to those in the training set. As the test set distribution becomes more bimodal it becomes *less* similar to a training set distribution, which is always unimodal. It is true that the results do become closer to the lower bound, but we attribute this to the fact that the L_2 lower bound becomes more accurate when the average size of the pieces is smaller, rather than because of an increase in performance.

Note the difference between the results of the evolved heuristics on test sets 1 and 3 to the results on test sets 2 and 4 (see figure 6). Test sets 1 and 3 have a mean piece size of 50. Sets 2 and 4 have a mean piece size of 33. The heuristics with memory perform noticably better than those without memory on sets 2 and 4, while the difference is not so obvious on sets 1 and 3. We attribute this to the ability of the heuristics with memory to learn to pack smaller pieces more effectively. There is potentially more opportunity to learn to leave space for the pieces which occur more frequently if more than two pieces fit into a bin on average.

At the beginning of the packing process, the memory of the heuristic is empty, as no pieces have yet been seen. As the packing progresses, the memory is filled up to a maximum of 100 pieces. Figure 7 shows the behaviour of the heuristics, on test set 10, in this period of time before the memory is completely full (100 pieces packed), and afterwards until the first 1000 pieces have been packed. The graph shows that at the start of the packing process, the heuristics with memory are worse than best-fit and the heuristics without memory. After approximately 50 pieces have been packed, the memory becomes useful and the pieces start to be packed more efficiently. By sacrificing performance at an early stage to build up a representative sample of the distribution of pieces, the heuristics with memory can perform better in the long term.

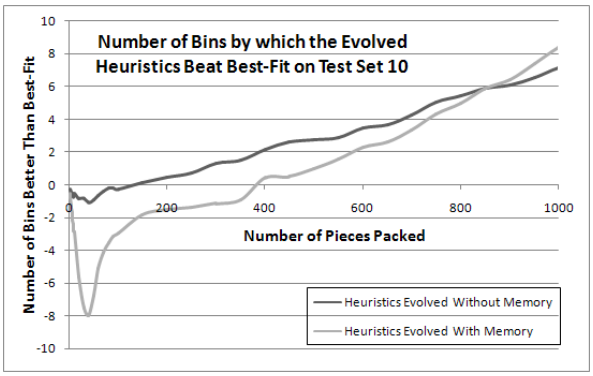

Fig. 7. A comparison of the packing behaviour of the evolved heuristics. The results are presented as a comparison against the performance of best-fit. Each line represents the number of bins used as the packing progresses, subtracted from the number of bins used by best-fit. While this graph is from test set 10, the results on the other test sets all show a similar pattern

IV. Conclusions

This work was motivated by the observation that previously evolved heuristics cannot use information about the pieces that have been seen so far during the packing process. Such information would be beneficial to a human performing the same task. For example, if the minimum piece size seen so far is 10, they would try to avoid putting a piece into a bin which leaves a gap of 9, as this gap will probably never be filled. Just as humans naturally make use of their experience, we investigated in this paper whether it was possible to evolve heuristics which were capable of the same ability, and, indeed, whether this resulted in better performance.

The inclusion of a memory that the heuristics can draw upon is a fundamental difference from previous work in this field, allowing automatically designed heuristics which can change their behaviour depending on their experience. Figure 7 shows clearly that the evolved heuristics perform better when their knowledge of the piece size distribution becomes more complete. Figure 6 shows that this ability to remember the pieces seen so far improves the performance of the evolved heuristics on every test set.

The difference between the heuristics with and without memory is more pronounced on the instances that are the most strongly multimodal, showing that they can make use of the information that the normal evolved heuristics do not have access to. We suggest that the heuristics with memory can determine that certain piece sizes will not appear in future, so the heuristics avoid leaving gaps of that size. The fact that heuristics with memory pack multimodal distributions better suggests that they must be using their memory to some degree to determine that some piece sizes are more likely to appear than others.

In conclusion, we have shown how genetic programming can be employed to evolve heuristics that can draw upon their 'experience', implemented as a memory of the piece sizes seen recently. We have shown that these heuristics can dynamically change their behaviour based on their experience. This is a fundamental difference over the fixed heuristic

strategies that have been evolved in previous work, and it results in heuristics that can pack more efficiently.

V. Future Work

The results of this study suggest potential areas for further research. The memory keeps a record of the last 100 pieces, but this value can be modified. It is currently unclear what the effects of changing the memory size would be. It is also unclear which memory terminals are the most important. We implemented a set of 8 memory terminals, but it will be interesting to investigate if the MIN, MAX, and AVE terminals are sufficient. We predict that the terminals providing information on the percentage of pieces of a certain size are more important than the MIN, MAX and AVE terminals, and therefore performance will decline if the full set is not included.

We also intend to investigate if the heuristics can perform well on a continuous packing line where the distribution changes over time, and where only a subset of bins can be open at a time. The ability to learn from experience would potentially be even more important in this situation, as different sections of the packing would ideally require different strategies from the same heuristic. The memory length would also be important in this situation, because if the distribution of pieces changes often, then older sections of the memory would become essentially obsolete. It would be interesting to see if the memory length itself could be evolved alongside the strategy which employs it.

Acknowledgment

This work was supported by EPSRC grant reference EP/D061571/1.

References

[1] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, "Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one," in *Proceedings of the 9th ACM Genetic and Evolutionary Computation Conference (GECCO'07)*, London, UK., July 2007, pp. 1559–1565.

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

[3] ——, "Lower bounds and reduction procedures for the bin packing problem," *Discrete Applied Mathematics*, vol. 28, no. 1, pp. 59–70, 1990.

[4] E. G. Coffman Jr, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: Combinatorial analysis," in *Handbook of Combinatorial Optimization*, D. Z. Du and P. M. Pardalos, Eds. Kluwer, 1998, pp. 151–207.

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

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

[7] 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*, vol. 3, no. 4, pp. 299–325, December 1974.

[8] C. Kenyon, "Best-fit bin-packing with random order," in *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, 1996, pp. 359–364.

[9] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, "The scalability of evolved on line bin packing heuristics," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'07)*, Singapore, September 2007, pp. 2530–2537.

[10] P. Ross, "Hyper-heuristics," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Boston: Kluwer, 2005, pp. 529–556.

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

[12] E. K. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyper-heuristic for timetabling and rostering," *Journal of Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

[13] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of Meta-heuristics 2nd Edition*, M. Gendreau and J.-Y. Potvin, Eds. Springer, to appear 2010.

[14] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward, "Exploring hyper-heuristic methodologies with genetic programming," in *Computational Intelligence: Collaboration, Fusion and Emergence*, C. Mumford and L. Jain, Eds. Springer, 2009, pp. 177–201.

[15] A. S. Fukunaga, "Automated discovery of local search heuristics for satisfiability testing," *Evolutionary Computation (MIT Press)*, vol. 16, no. 1, pp. 31–1, 2008.

[16] M. B. Bader-El-Din and R. Poli, "Generating SAT local-search heuristics using a gp hyper-heuristic framework," in *LNCS 4926. Proceedings of the 8th International Conference on Artifcial Evolution*, October 2007, pp. 37–49.

[17] C. D. Geiger, R. Uzsoy, and H. Aytug, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *Journal of Scheduling*, vol. 9, no. 1, pp. 7–34, 2006.

[18] E. K. Burke, M. R. Hyde, and G. Kendall, "Evolving bin packing heuristics with genetic programming," in *LNCS 4193, Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN'06)*, T. Runarsson, H.-G. Beyer, E. Burke, J. J.Merelo-Guervos, D. Whitley, and X. Yao, Eds., Reykjavik, Iceland, September 2006, pp. 860–869.

[19] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, "A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics," *IEEE Transactions on Evolutionary Computation (accepted, to appear)*, 2010.

[20] S. Allen, E. K. Burke, M. R. Hyde, and G. Kendall, "Evolving reusable 3d packing heuristics with genetic programming," in *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO '09)*, Montreal, Canada, July 2009, pp. 931–938.

[21] H. Terashima-Marin, E. J. Flores-Alvarez, and P. Ross., "Hyper-heuristics and classifier systems for solving 2d-regular cutting stock problems," in *Proceeedings of the ACM Genetic and Evolutionary Computation Conference (GECCO'05)*, Washington, D.C. USA, June 2005, pp. 637–643.

[22] H. Terashima-Marin, C. J. F. Zarate, P. Ross, and M. Valenzuela-Rendon, "A GA-based method to produce generalized hyper-heuristics for the 2d-regular cutting stock problem," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'06)*, Seattle, WA, USA, July 2006, pp. 591–598.

[23] H. Terashima-Marin, C. J. F. Zarate, P. Ross, and M. Valenzuela-Rendon., "Comparing two models to generate hyper-heuristics for the 2d-regular bin-packing problem," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'07)*, London, England, July 2007, pp. 2182–2189.

[24] H. Terashima-Marin, P. Ross, C. J. Farias-Zarate, Lopez-Camacho, and Valenzuela-Rendon, "Generalized hyper-heuristics for solving 2d regular and irregular packing problems," *Annals of Operations Research*, vol. Available online November 16, 2008.

[25] 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)*, New York, NY., 2002, pp. 942–948.

[26] 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)*, Chicago, Illinois, 2003, pp. 1295–1306.

[27] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Boston, Massachusetts: The MIT Press, 1992.