The Scalability of Evolved On Line Bin Packing Heuristics

E. K. Burke, M. R. Hyde, G. Kendall, J. R. Woodward

Abstract— **The on line bin packing problem concerns the packing of pieces into the least number of bins possible, as the pieces arrive in a sequential fashion. In previous work, we used genetic programming to evolve heuristics for this problem, which beat the human designed 'best fit' algorithm. Here we examine the performance of the evolved heuristics on larger instances of the problem, which contain many more pieces than the problem instances used in training. In previous work, we concluded that we could confidently apply our heuristics to new instances of the same class of problem. Now we can make the additional claim that we can confidently apply our heuristics to problems of much larger size, not only without deterioration of solution quality, but also within a constant factor of the performance obtained by 'best fit'.**

Interestingly, our evolved heuristics respond to the number of pieces in a problem instance although they have no explicit access to that information. We also comment on the important point that, when solutions are explicitly constructed for single problem instances, the size of the search space explodes. However, when working in the space of algorithmic heuristics, the distribution of functions represented in the search space reaches some limiting distribution and therefore the combinatorial explosion can be controlled.

I. Introduction

Recent years have seen the emergence of a more general type of search algorithm, hyper-heuristics [1], [2], where the search space is a space of heuristics, rather than the space of solutions themselves. This leads to more general solution methods, and we wish to examine the scalability of these solutions in this paper. Hyper-heuristics are heuristics which choose *"between a set of low-level heuristics, using some learning mechanism"*[3]. i.e. a hyper-heuristic manages a fixed set of heuristics.

One of the motivations of hyper-heuristics [1], [2] is to "raise the level of generality at which optimisation systems can operate", providing a search methodology which will deliver solutions which are *"good-enough soon-enough cheap-enough"* [2]. While this 'off the peg' approach is unlikely to produce solutions which are as good at those produced by a 'tailormade' problem specific method, the goal is to underpin decision support systems which can be applied to a broader range of problems than is possible today.

Hyper-heuristics can also be used to produce new heuristics, which are not very dependent upon the problem at hand. Previous work [4], [5] demonstrated that this is achievable. In this paper, we show that the heuristics produced can be applied to larger problems than those used in the training phase, without loss of performance.

E. K. Burke, M. R. Hyde, G. Kendall, J. R. Woodward are with the Automated Scheduling, Optimisation and Planning (ASAP) research group School of Computer Science and Information Technology, University of Nottingham, Jubilee Campus, Nottingham NG8 2BB, UK. (phone: +44 (0)115 951 4206; fax: +44 (0)115 951 4249; email: (ekb,gxk,mvh,jrw)@cs.nott.ac.uk).

The approach described in this paper is different to many other evolutionary approaches to solving combinatorial problems. Typically, with conventional approaches a problem is solved directly (see [6]). For example, if we were solving an instance of the Travelling Salesman Problem, a candidate solution would consist of a list of the cities to be visited and the search method (e.g. genetic algorithms) would directly manipulate a representation of the route (i.e. the route is encoded in the genotype).

Our goal, in this paper, is to present a system which can automatically generate heuristics which can be used for a class of problems. The difference between conventional methods and evolving heuristics can be summarized by the proverb,

"*Give a man a fish and he will eat for a day, teach a man to fish and he will eat for a lifetime*".

One of the major advantages of producing a general heuristic is that it can be applied to additional problems without the need for further parameter tuning. Secondly, if we want to tackle a really large instance of a problem, then if we were using the direct approach we would have to deal with large encodings of the potential solutions and the search space would explode. However, if we are searching the space of heuristics, then the search space will not explode as we are dealing with general solution methods. It is the aim of this paper to consider how the behaviour of evolved heuristics scales to larger problems.

II. Related Work

We will briefly discuss some examples of previous hyper-heuristic methods that have appeared in the literature. Two hyper-heuristic methods have been tested on the one dimensional bin-packing problem, a learning classifier system [7] and a genetic algorithm [8]. Simulated annealing is used as a hyper-heuristic in [9] for the shipper rationalisation problem. A case based reasoning hyper-heuristic is used in [10] for both exam timetabling and university course timetabling. Three new hyper-heuristic architectures are presented in [11], which treat mutational and hill climbing low-level heuristics separately. A graph based hyper-heuristic is presented in [12]. In [13], a tabu search hyper-heuristic is presented and evaluated upon a nurse scheduling problem and a university course timetabling problem. A choice function has also been employed as a hyper-heuristic, to rank the low-level heuristics and choose the best one [14], and a distributed choice function hyper-heuristic is presented in [15]. The choice function considers the recent effectiveness of each

heuristic and each pair of heuristics, and also considers the time since the heuristic was last called.

The bin packing problem is NP-Hard [16], so a polynomial-time exact algorithm is unlikely to be found for the general case [17]. Several different heuristics have been developed in the literature. A number of examples of heuristics used in the on line bin packing problem are described below:

Best Fit [18]. Puts the piece in the fullest bin that has room for it. Opens a new bin if the piece does not fit in any existing bin.

Worst Fit [17]. Puts the piece in the emptiest bin that has room for it. Opens a new bin if the piece does not fit in any existing bin.

Almost Worst Fit [17]. Puts the piece in the second emptiest bin if that bin has room for it. Opens a new bin if the piece does not fit in any open bin.

Next Fit [19]. Puts the piece in the right-most bin and opens a new bin if there is not enough room for it.

First Fit [19]. Puts the piece in the left-most bin that has room for it and opens a new bin if it does not fit in any open bin.

A. GP applied to problems of different size

A number of problems of different size have been studied in the GP literature, for example the even parity problem and the lawn mower problem [20], [21]. Typically, as the size of the problem grows, the size of the solution grows and, correspondingly, the time taken to find the solution grows. Another example of a scalable problem includes image recognition problems. For example, we could consider the problem of identifying a certain image on a certain fixed sized image array (e.g. 100 by 100 pixels), but this could be generalised to the problem of identifying the certain image on an n by n array. In fact many problems can be considered to come from a more general incarnation, where the dimension could be size, or some other parameter which describes the problem class.

In terms of representation, standard GP has difficulties with problems of different size. Teller [22] states that the language used by standard GP is not powerful enough to express many algorithms. In particular, *'There is no mechanism for variable length strings to be shown to the function and no way for the function to iterate an arbitrary number of times'*. A more expressive representation is needed if we are to tackle generalisations of variable size problems. There may be other benefits too by looking at small instances of problems before looking at larger instances (i.e. the small problems may contain enough information to generalise from, without considering larger instances of the problem, hence much processor time can be saved).

In terms of standard GP, two instances of problems of different sizes are considered as *different* problems i.e. we may evolve a solution to the even-3-parity problem *or* the even-4-parity problem. In the former case, we would have 3 variables in the terminal set, and in the latter case 4. This is interesting as many people would consider these to be *instances* of the same problem as the nature of the underlying problem is the same in both cases (i.e. is the number of true bits even?). In the arena of GP, problem variables are often represented explicitly in the solution, a better approach might be to have the ability to address a general variable (e.g. the ith bit).

Miller et. al. [23] evolve digital designs of arithmetic functions (addition and multiplication) and argue that *'by studying the evolved design of gradually increasing scale, one might be able to discern new, efficient, and general principles of design'*. They pose the central question in the paper, *'Can we by evolving a series of subsystems of increasing size, extract the general principle and hence discover new principles?'* While we agree with this approach, it may be possible however, to evolve a general algorithm rather than having a human study the evolved designs.

Brave [24] evolves solutions to a planning problem using automatically defined functions with a restricted form of recursion. He shows that the effort required to find solutions with automatically defined functions and recursion remains constant with the problem size. Basic GP scales exponentially and automatically defined functions scale linearly at best, which is consistent with Koza's [21] results for basic GP and GP+automatically defined functions on the lawn mower problem.

We may not need or be interested in a general solution to a problem, we may simply want a solution to a fixed size instance of the problem. For example we may be interested in the solution of the lawn mower problem for a lawn of size 10000 by 10000, or the even-10^6-parity problem. In these cases it would be easier to evolve a general solution to the problems, rather than evolving a specific solution. Also tackling smaller instances of the problem makes sense (i.e. it is the same underlying problem and we can learn the solution to the underlying problem before tackling larger instances). Yu [25] (page 361, section 6.2) includes in her test set all cases of even-2-parity and even-3-parity problems (i.e. $2^2 + 2^3 = 12$ test cases), and this is enough to evolve a general solution to the even parity problem.

B. A fundamental difference in the search spaces

There is an important difference between the search space associated with direct methods, and the search space associated with evolving heuristics. With direct methods, as we tackle larger problems, the size of the genotype will increase in direct correspondence with the size of the problem instance. For example, tackling an n-city instance of the travelling salesman problem, will involve a search space containing candidate individuals of length n, leading to *a combinatorial explosion*.

If we are evolving a heuristic for the problem, as we are evolving a general algorithm, the size of the heuristic will not depend on n (the size of the problem). Hence in this case, we could evolve a heuristic for a small or large instance of the problem, but the number of evaluations will be independent of the problem size.

TABLE I

INITIALISATION PARAMETERS OF EACH GP RUN

Parameter	Value
Population size	1024
Maximum generations	50
Crossover probability	0.9
Reproduction probability	0.1
Tree initialisation method	Ramped half-and-half
Selection method	Tournament selection, size 7
Function set	$+, -, *, /$
Terminal set	F, E

Also, another fundamental difference between these search spaces is pointed out by Langdon [26]. Langdon investigates the limiting frequency with which functions are represented in various search spaces constructed for GP. He is concerned with the limiting distribution of functions. In some cases he proves that the limiting distribution of functions does not change above some threshold, provides empirical evidence in other cases, and makes conjectures for further cases.

III. THE ON LINE BIN PACKING PROBLEM

Bin packing problems consist of taking a number of pieces and packing them into a number of bins. The aim is to pack the pieces into as few bins as possible. The most popular version of the problem is the off line version, where a *set* of pieces is known in advance, and this set can be packed into the bins in any order. We tackle the on line version, where a *sequence* of pieces is to be packed. Not only must the sequence of pieces be packed in the order they are presented, but we do not know the sequence beforehand (i.e. we do not have the ability to look ahead). In this paper, bins are all of the same size (150 units), and piece sizes are drawn uniformly from a distribution of size 20 to 100 units.

IV. EXPERIMENTS

In this section we explain how we apply GP to our problem. We describe how the GP system is set up (this is fairly standard), how the trees produced by GP are applied to instances of the problem, and how the fitness is calculated for multiple problem instances. The parameter settings are presented in table I.

A. Parameter Settings

We used Sean Luke's Java-based Evolutionary Computation Research System ($http://www.cs.gmu.edu/eclab/projects/ecj/$) to implement our GP system. The contribution of this paper is not a new innovation regarding GP, but an investigation into the scalability of the evolved heuristics applied to the on line bin packing problem.

In a single run, a heuristic is evolved on 20 problem instances consisting of 100, 250 or 500 pieces. We evolve 30 heuristics (i.e. 30 runs) for each problem size, thus we evolve 90 heuristics all together. These 90 heuristics are then applied to 20 independent problem instances consisting of 100000 pieces each.

The quantity we are trying to minimize is the number of bins used to legally contain the pieces. There are two components to our fitness function. A natural measure is used to assign a fitness equal to the number of bins, to the given heuristic. This seemingly obvious fitness function is not a particularly good evolutionary driver on its own, as many heuristics can use the same number of bins, and so there is little evolutionary pressure to encourage further improvement. Hence, we need a way of differentiating between heuristics which produce packing configurations consisting of the same total number of bins. The following fitness function (shown in equation 1), does just this, where:
n = number of bins used, F_i = fullness of bin i, and C = bin capacity.

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

We can now combine these two fitness functions. We use the total number of bins as our fitness function, and if two heuristics obtain the same value, then we use the fitness function described in equation 1 to differentiate between two heuristics which use the same number of bins.

B. Applying the heuristic

An evolved heuristic is used to choose in which bin to place the current piece under consideration, by scoring each bin and placing the piece in the bin with the maximum score. An evolved algorithm is applied to the current set of bins. This set always includes an empty bin. The algorithm always has the choice of putting the current piece in the empty bin. If a piece is placed in the empty bin, a new empty bin is added to the set, maintaining the condition that there is always an empty bin available. The current piece is placed in the bin which receives the maximum score according to the evolved heuristic. A heuristic can place a piece in a bin causing the size of its contents to exceed its capacity. We do allow this to occur in the evolution; however such heuristics receive a large penalty fitness. Heuristics which produce illegal solutions typically die out in the first couple of generations, though, of course, there is always a small probability that illegal heuristics will form in mature populations.

C. Note on the terminal set

In our original work [4], we used a terminal set consisting of the size of the current piece, S, the capacity of the bins, C, and the fullness of the current bin being considered by the heuristic, F, i.e. S, C, F. However, upon reflection, we came upon the following improvement. The fullness and capacity of a bin can be combined $(C - F)$ to give the emptiness, E, of a bin, which is the space remaining in the bin. When there are no other pieces in the bin, $E = C$. The quantity $(E - S)$ would be the space remaining in the bin if the current piece were placed in the bin. In a sense, the fullness and capacity are irrelevant; it is only the remaining space in the bin E, and the size of the current piece which are necessary to describe the state of a given bin. While this set of terminals is slightly less expressive, it is still adequate to express solutions to the problem. Also, as we have reduced the number of variables

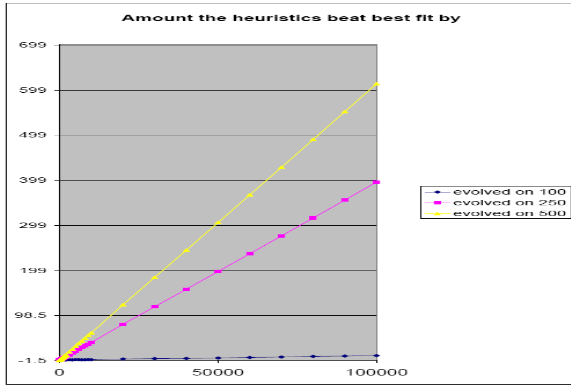

Fig. 1. The horizontal axis shows the number of pieces packed. The vertical axis shows the number of bins that a heuristic beats 'best fit' by. This is done for heuristics evolved on problem instances of three different sizes (containing 100, 250 or 500 pieces). This horizontal scale goes up to 100000 pieces, which is much larger than the number of pieces contained in the problem instances used in the training phase. These results are averaged over 30 heuristics over 20 problem instances.

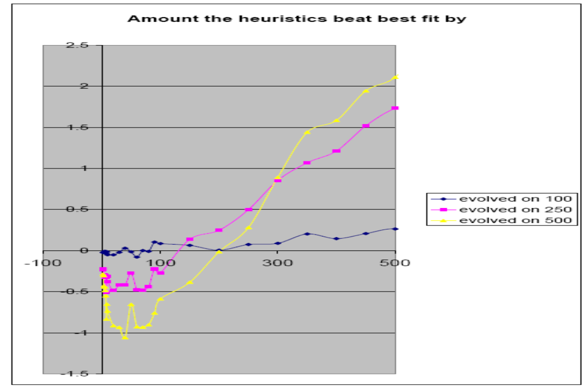

Fig. 2. The horizontal axis shows the number of pieces packed. The vertical axis shows the number of bins a heuristic beats 'best fit' by. This is done for heuristics evolved on problem instances of three different sizes (containing 100, 250 or 500 pieces). Note that the horizontal scale covers the size of the training problem instances.

from 3 to 2, this allows us to plot graphs of the heuristics' performance, which are presented later.

D. Note on protected divide

Often in GP, division is included in the function set which can cause problems if the denominator is zero. Protected division is used where 1 is returned if the denominator is zero. This causes a slight problem with this application as when $(E - S)$ is zero (i.e. the piece fits perfectly) only a small value is returned and the piece is not likely to be placed in that bin. To avoid this problem we set the denominator to 0.5. Hence, our protected divide function returns a number larger than if the denominator was 1 or greater. In other words, using the standard protected divide function, as the denominator gets smaller, the expression gets larger, except when the denominator hits zero and a small value is returned. Hence, this method of protecting division is a little unnatural. We decided to return double the numerator when the denominator hits zero.

V. Results

A. Performance outside training range

Figure 1 shows how heuristics evolved on small problems (either 100, 250 or 500 pieces) perform on much larger problems. The horizontal axis is the number of pieces in the problem instance. The vertical axis is the number of bins that the evolved heuristic beats the 'best fit' heuristic by (e.g. at 100000 pieces, heuristics evolved on 500 pieces beat 'best fit' by about 600 pieces). There are a number of observations which can be made. Firstly, the heuristics do scale, and there is not some catastrophic deterioration in performance on larger problems. Secondly, we can predict what the number of bins will be relative to the 'best fit' heuristic. Thirdly, the larger the number of pieces in the training set the heuristic was evolved on, the more it beats 'best fit' by.

B. Statistics

Table I	H100	H250	H500
100	0.427768358	0.298749035	0.140986023
1000	0.406790534	0.010006408	0.000350265
10000	0.454063071	2.57785E-07	9.65298E-12
100000	0.271828318	1.37522E-25	2.78293E-32

Table I shows statistics for the results produced. The 30 heuristics evolved on each problem instance size, are tested on 20 independent problems, and the number of bins used at 100, 1000, 10000, 100000 pieces are recorded (i.e. 20 values for each). We then compare each of these 20 values with 20 values obtained with the 'best fit' heuristic. Columns 1, 2 and 3 show the values for the three training set sizes. Rows 1 to 4 show the values for the number of pieces in the validation problem instances. We have recorded the values for the one tailed student t-test, which show the probability that the two distributions come from the same distribution. Generally, as the number of pieces in the instance increases, our confidence that the two distributions are different grows. Also, as the size of the training instances grows, our confidence that the two distributions differ increases.

C. Performance over training range

Figure 2 has the same axes as figure 1, but we are zooming into the region of the graph which contains the results for problems with pieces up to 500 (i.e. the maximum range on which we trained). We can make the following observation. For heuristics trained on problems consisting of 100 pieces, 'best fit' beats the heuristic up to around 50 pieces (i.e. the line is below the x axis). Past this point, the line recovers from its poor initial start, and remains positive. For heuristics trained on problems consisting of 250 pieces, 'best fit' out performs the heuristics up to around 125 pieces, but after this point the heuristics starts to pack pieces more efficiently and at the 250 piece stage the heuristics beat 'best fit'. A similar pattern of behaviour is observed with the heuristics trained with problems consisting of 500 pieces. Until the 200 piece mark, 'best fit' starts by packing more efficiently,

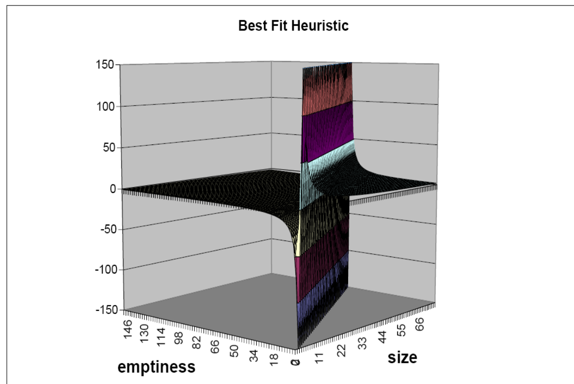

Fig. 3. The horizontal axes show the emptiness of a bin varying from 0 to 150, and the current piece size varying from 0 to 70. The vertical axis is the value returned by the 'best fit' heuristic. The piece is placed in the bin which corresponds to the maximum value.

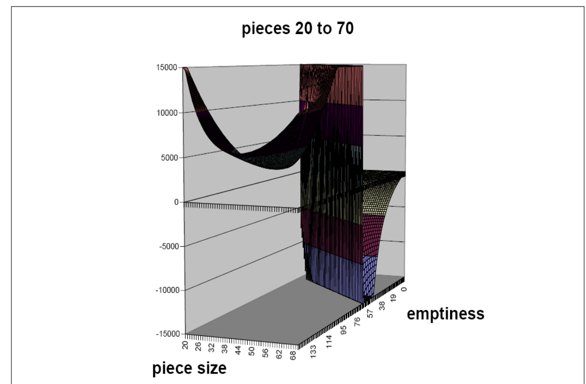

Fig. 4. This is an evolved heuristic. Note the axes are the same as in figure 3, but is rotated slightly. Notice that this is very similar to the plot of the 'best fit' (figure 3), but differs significantly in one corner of the plot.

then the heuristics overtake it. We can make the qualitative observation that if a heuristic is trained on a sequence of pieces, typically for the first half of a sequence, 'best fit' will out perform it, but in the second half the heuristic catches up and overtakes' 'best fit'.

It is perhaps worth drawing an analogy here between these heuristics and competitors in a running race. Often the strategy employed by athletes is to hold back initially and in the final stages accelerate and pass the front runners. Often the people who lead for most of the race do not win and are used by the eventual winners to "set the pace".

D. Best Fit

In this subsection, we explain the 'best fit' algorithm, examine a graph of 'best fit' (figure 3), and explain intuitively why it is a good heuristic. The 'best fit' algorithm fits the current piece under consideration, into the smallest gap in a bin into which it will fit. It the piece does not fit into a bin, then a new bin is opened and the piece is placed in it. This heuristic has the property that it never opens a new bin unnecessarily. If the piece can go into an existing bin, then 'best fit' will place it without opening a new bin. 'Best fit' then, in some respects is a very efficient strategy.

In our set up, the heuristic (human or evolved) is applied to each bin and the highest scoring bin is the bin in which we place the piece. This function is expressed algorithmically as $C/(E-S)$ in this framework. This function is plotted in figure 3. The vertical scale is essentially irrelevant as we take the maximum value of the function, so multiplying by a factor will make no difference. The space is divided into two by a vertical plane. Notice that when $E-S<0$, the plot is negative (indicating an illegal solution). When $E-S$ is small, the larger the value 'best fit' assigns to that bin-piece combination. When the heuristic is presented with a set of n bins and a current piece to be placed in the bins, each bin will give rise to a point on the plot, and we choose the point which corresponds to the largest function value, and this is the bin in which the piece is placed. It is worth taking a minute to examine the plot as this will be compared to our best evolved heuristic.

E. Best Evolved Heuristic

In figure 4 we plot the function values of our best observed heuristic. We now outline this plot. It is rotated relative to the 'best fit' plot (figure 3), but very similar features can be seen. A vertical plane divides the space into two. Behind the plane, all of the function values are negative. In front of the vertical plane the function diminishes, very much like the 'best fit' heuristic, except in the region when piece size is small and the emptiness is large, and the function value curls upwards. There is also an interesting ridge running approximately parallel to the vertical plane. We do not know yet if this ridge is of significance or not.

This function is very similar to the function which expresses 'best fit', except for the case when a piece is small and a bin is empty. It will prefer to put small pieces in very empty bins, and in a sense is doing something similar to the "worst fit" heuristic i.e. it is placing a small piece into a bin with a large gap. This may seem counterintuitive when compared to the rather intuitive 'best fit' approach. This may be explained by the following example. We may have the situation where a gap of size 30 exists, and a piece of size 20 arrives. Best fit may understandably place the piece in this bin. However, if we know that we still have many more pieces to fit, then it is worth taking the less myopic approach and waiting for a piece which is a better fit (e.g. between size 25 and 30), and places the piece of size 20 into a fresh bin.

As this algorithm tends to put small pieces in very empty bins, it probably has a larger distribution of gaps in which to fit future pieces. 'Best fit' is rather greedy in this respect. A better long term approach would be to build up a set of gaps which can best accommodate incoming pieces, with the expectation that pieces which will fit the gaps will arrive eventually. Our algorithm is essentially anticipating that a better fitting piece will come along before the last piece in the sequence arrives.

As we showed in previous work [5], we produced heuris-

tics which were tuned to the problem class they were exposed to in the training phase i.e. heuristics evolved on problems consisting of pieces of a given distribution performed better on problems which consisted of pieces with the same distribution of sizes. Thus GP is reacting to the distribution of pieces in the problem and producing a function which contains a predictive quality. We can now also claim GP is reacting to the number of pieces in the problem.

VI. Discussion

If we were to tackle this problem in a theoretical framework, one approach would be to look at the expectation of the number of bins given the probability distribution over the pieces sizes that we are likely to pack. We are effectively planning under uncertainty as we do not know what the sequences of pieces are beforehand. Thus we could only give probabilistic responses to the question as to the bin in which we should place the current piece. In our current framework, we place the piece in the bin which scores maximally according to the heuristic. Evolved heuristics which behave legally (i.e. do not over fill a bin), will return a set of positive values for the potential bins. Thus (after normalizing) we could interpret the output of the heuristic as a probability, and we are choosing the bin which has the maximum probability associated with it.

Thus, currently we are just placing a piece in a bin at each stage. If we interpret the value of the heuristic as a probability, then we will also be able to make claims about how certain we are about the action of placing the current piece in the selected bin. While this may not be of direct interest in the abstract version of the bin packing problem, it may well be of interest in certain real world applications.

It is clear from figure 1 that there exist some soft thresholds regarding the size of the training instances. Firstly, there appears to be a lower threshold below which we cannot induce a heuristic algorithm to beat 'best fit'. This is around 100 pieces, as is evident as the trace is just above the horizontal axis. In this case, we can learn a bin packing strategy which just beats 'best fit'. Thus, for the distribution of piece sizes used in this paper and the bin capacity stated, there exists a lower threshold at around 100 pieces.

Secondly, as we increase the number of pieces in the training data to 250 and 500, we see an increase in performance in the evolved heuristic. This increase in performance will increase with the number of pieces in the training sets, but as the number of pieces increases we will start to approach the theoretical optimum (i.e. the sum of the sizes of all of the pieces divided by the bin capacity) and we will see little further improvement in performance as we increase the number of pieces.

The on line bin packing problem has a strict mathematical definition, where we are not told in advance how many pieces there are in a particular problem instance. In some respects this detail may appear not to be important. However, as we see in the results presented in this paper, if we are near the start of the sequence of pieces to be packed, then we can pack pieces more loosely as potentially better fitting pieces may exist later in the sequence. If we are towards the end of the sequence then, this signals that we need to start packing pieces more tightly.

The evolved heuristic returns a value when it is presented with a piece size S and bins with emptiness E. It does not have memory [27], and it can only examine one bin at a time. Hence, the value returned by the algorithms is independent of the emptiness values of the other bins, and independent of what position the piece is in the sequence (i.e. it could be the first or last piece in the sequence). However, the GP system seems to have learnt something about the state of the bin packing system (i.e. as seen in figure 2), the heuristic seems to know that the last pieces are approaching without having explicit access to state information (e.g. the current piece is the last piece).

How can our heuristic appear to do this without having explicit access to state information. When a set of bins is evaluated for their suitability for the current piece, this corresponds to a set of points on the surface of the plot in figure 4. As the number of bins increases, then the number of points on the surface of the plot will increase. As the number of bins increases, then the "competition" between them increases (i.e. a bin which may have won in the past may not win now as there are more suitable gaps available). Hence, in some respects there is at least a memory effect.

The main motivation of the approach presented is to evolve an algorithm, which can be reused on future problem instances. However, in view of these results, an interesting approach to tackling a large single instance of a problem may be to evolve an algorithm on a number of small problem instances (from the same problem class) and then use the resulting algorithm to tackle the larger single instance of the problem. For example, given that we had a bin packing problem with a large number of pieces to pack, in the long term, it may be better to evolve a heuristic on problems with a small number of pieces and apply the resulting heuristic to the larger problem.

VII. Further work

In earlier work [4], we allowed our heuristic to examine each bin in turn, and it placed the current piece in the *first* bin with a positive value according to the evolving heuristic. This allowed us to evolve heuristics whose performance was comparable with the 'first fit' heuristic [4]. In further work, we allowed the heuristic to scan all of the bins and place the piece in the bin which receives the *maximum* score [5]. This allowed us to evolve heuristics whose performance was comparable with 'best fit'. However, this second approach (which we are also using in this paper), does not allow us to express the first fit algorithm or the types of algorithm found in the first case. Similarly, the first positive scoring bin approach does not allow us to express the 'best fit' heuristic. It would be interesting to construct a search space which was capable of expressing 'first fit' type heuristics and 'best fit' type heuristics.

Typically, in real world situations involving the bin packing problem, the number of pieces for a particular instance

of the problem may not be known in advance. However, over time as we see more instances of the problem, a distribution over the number of pieces will begin to emerge (i.e. we are sampling this distribution). This will provide an additional source of information that GP will be able to make use of, which we expand upon below.

If we know how many pieces there are to be packed, we can use this information to improve our bin packing heuristics. An outline for a three stage approach is as follows. In the first stage, initial pieces are packed (this would be around 100 pieces, as observed earlier in the paper, which is just enough to induce bin packing behaviour). In this stage, we attempt to build up the contents of the bins. We then enter the second stage where additional pieces arrive and are packed, essentially under the assumption that there are many more pieces to arrive (i.e. we do not have to employ a greedy algorithm, but can open fresh bins which we expect will get better filled in the long term, rather than forcing in pieces which fit but leave a gap). In effect, we are maintaining a good distribution of gaps in the bins in which to receive future pieces. In the final stage, we are nearing the end of the sequence so it makes sense not to open new bin if it can be avoided. It would be interesting to see if we could evolve an algorithm which learns to switch behaviour, depending on whether the current piece is at the start of the sequence or near the end of the sequence.

In this work, and previous work[4], we have generated pieces using uniform probability distributions. 'Best fit' performs well when the piece size distribution is uniform, however this may not be the case with a very rugged distribution of pieces. One of the strengths of using GP to evolve heuristics is that we can now tackle more general problem classes where the probability distribution is far from uniform.

In this work, the fitness function is based on the number of bins. It would be interesting to use a multi objective approach to plot the Pareto front of fast and near optimal bin packing algorithms allowing the investigation of the trade off i.e. one axis is the number of bins a heuristic uses, the second axis is the speed at which it executes. We make the conjecture that the Pareto front will be a convex hull.

Also, if we were particularly interested in speed, then the type of wrapper used to apply the GP heuristic to the bin packing problem will become more important. For example, if we used a first fit type wrapper (where the piece is placed in the first bin which receives a positive score), this will typically run faster than a 'best fit' type wrapper (where all of the bins are examined). A search space which includes both of these types of wrappers, would be of particular interest in this case.

In our current implementation, the evolved heuristic is applied to all of the bins. Over time, as more pieces are placed in bins, and more bins are opened, the heuristic will take longer and longer to apply. There are two things we can do to improve the run time of the algorithm. Firstly, full bins can be removed from the computation (a bin is considered full if the emptiness is less that the size of the smallest piece in the distribution). Secondly, in the portion of the graph we are interested in (see figure 4), the profile of the surface is approximately unimodal (i.e. has a single trough running through the centre). If we can approximate this with a unimodal function, then the bins can be ordered and the bin which scores the maximum value according to the heuristic can be found by bisection rather than considering each of the bins in turn. Of course, if we extend the wrapper we apply to the heuristics, it is possible that GP could discover this.

We are currently investigating the use of GP to evolve heuristics for the 2D cutting and packing problem. We are also extending our implementation of the on line bin packing problem, to the off line bin packing problem. In these related problem domains, we expect similar results (e.g. in terms of human competitiveness and robust performance with respect to scalability). It is our intention to develop principles which apply across problem types, rather than developing specialised techniques which can only be applied to a narrow category of problem.

VIII. Summary and Conclusions

We have evolved heuristics for the on line bin packing problem. We are given a sequence of pieces which must be packed in the order they arrive. A heuristic is applied to each of the bins and the current piece is placed in the bin which receives the maximum score according to the heuristic. In earlier work [4] we produced heuristics comparable to the human designed 'best fit' heuristic. In the current paper, we examine how the evolved heuristics scale to problem instances which contain many more pieces than were contained in the problem instances used in the training phase. In general, we showed that the evolved heuristics do scale as one might expect, without deterioration in performance and with a constant multiplicative improvement over the performance obtained using 'best fit'.

We also demonstrated an interesting memory effect. While heuristics do not have access to the number of pieces in the sequence, or what position the current piece is in the sequence, the heuristics seem to learn this and use it to their advantage. While the number of pieces is not specified in the general manifestation of the bin packing problem, this information will be available in many industrial applications, and therefore it is worth giving GP access to this in order to evolve more effective heuristics.

Finally, we pointed out the importance of evolving general heuristics, rather than solutions to specific problem instances. We also highlighted the fact that the search spaces associated with direct methods and methods involving the evolution of heuristics, are fundamentally different and this difference can be exploited to avoid (or at least reduce) the effects of the combinatorial explosion.

There are some interesting and healthy conclusions which can be drawn from the observations made in this paper.

- If a heuristic is trained on problem instances containing n pieces, it can perform better than 'best fit' on problem instances which contain more than n pieces.

- It will perform worse than 'best fit' on problems which contain pieces less than approximately $n/2$ pieces
- The larger the number of pieces in the instances in the training set, the better the scaling performance
- There exists a lower threshold (just below 100 pieces). below which it was not possible to evolve behaviour competitive with 'best fit'. This threshold will in general vary according to the distribution of pieces and their sizes relative to the bin capacity.
- As problem instances with more pieces are used in the training phase, the performance of the evolved heuristic increases. There will exist an asymptotic threshold above which little gain in performance can be seen.

REFERENCES

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

[2] Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In Glover, F., Kochenberger, G., eds.: Handbook of Meta-heuristics. Kluwer (2003) 457–474

[3] Soubeiga, E.: Development and Application of Hyperheuristics to Personnel Scheduling. PhD thesis, Univesity of Nottingham, School of Computer Science (2003)

[4] Burke, E.K., Hyde, M.R., Kendall, G.: Evolving bin packing heuristics with genetic programming. In Runarsson, T.P., Beyer, H.G., Burke, E., Merelo-Guervos, J.J., Whitley, L.D., Yao, X., eds.: Parallel Problem Solving from Nature - PPSN IX Springer Lecture Notes in Computer Science. Volume 4193 of LNCS., Reykjavik, Iceland, Springer-Verlag (2006) 860–869

[5] Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.R.: Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In Lipson, H., Thierens, D., eds.: Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, UK, July 7-11, 2007, ACM (2007)

[6] Sastry, K., Kendall, G., Goldberg, D.: Genetic algorithms. In Burke, E.K., Kendall, G., eds.: Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. Springer, Boston (2005) 97–125

[7] Ross, P., Schulenburg, S., Marín-Blázquez, J., Hart, E.: Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In Langdon, W.B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M.A., Schultz, A.C., Miller, J.F., Burke, E., Jonoska, N., eds.: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, Morgan Kaufmann Publishers (2002) 942–948

[8] Ross, P., Marin-Blazquez, J.G., Schulenburg, S., Hart, E.: 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) 1295–1306

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

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

[11] Ozcan, E., Bilgin, B., Korkmaz, E.E.: Hill climbers and mutational heuristics in hyperheuristics. In Runarsson, T., Beyer, H.G., Burke, E., J.Merelo-Guervos, J., Whitley, D., Yao, X., eds.: Lecture Notes in Computer Science, Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006). Volume 4193., Reykjavik, Iceland (2006) 202–211

[12] Burke, E.K., McCollum, B., Petrovic, A.M.S., Qu, R.: A graph-based hyper heuristic for timetabling problems. European Journal of Operational Research **176** (2007) 177–192

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

[14] Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In Burke, E.K., Erben, W., eds.: Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT 2000), Springer Lecture notes in Computer Science. Volume 2079., (Konstanz, Germany)

[15] Rattadilok, P., Gaw, A., Kwan, R.: Distributed choice function hyper-heuristics for timetabling and scheduling. In Burke, E., M.Trick, eds.: Practice and Theory of Automated Timetabling V, Springer Lecture notes in Computer Science. Volume 3616. (2005) 51–67

[16] Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)

[17] Coffman Jr, E.G., Galambos, G., Martello, S., Vigo, D.: Bin packing approximation algorithms: Combinatorial analysis. In Du, D.Z., Pardalos, P.M., eds.: Handbook of Combinatorial Optimization. Kluwer (1998)

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

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

[20] Koza, J.R.: Genetic programming II: automatic discovery of reusable programs. MIT Press, Cambridge, MA, USA (1994)

[21] Koza, J.R.: Scalable learning in genetic programming using automatic function definition. In: Advances in genetic programming. MIT Press, Cambridge, MA, USA (1994) 99–117

[22] Teller, A.: Turing completeness in the language of genetic programming with indexed memory. In: Proceedings of the 1994 IEEE World Congress on Computational Intelligence. Volume 1., Orlando, Florida, USA, IEEE Press (1994) 136–141

[23] Miller, J.F., Job, D., Vassilev, V.K.: Principles in the evolutionary design of digital circuits - part i. Genetic Programming and Evolvable Machines **1**(1/2) (2000) 7–35

[24] Brave, S.: Evolving recursive programs for tree search. In Angeline, P.J., E., K., eds.: Advances in Genetic Programming 2. MIT Press (1996)

[25] Yu, T.: Hierarchical processing for evolving recursive and modular programs using higher-order functions and lambda abstraction. Genetic Programming and Evolvable Machines **2**(4) (2001) 345–380

[26] Langdon, W.B.: Scaling of program tree fitness spaces. Evolutionary Computation **7**(4) (1999) 399–428

[27] Teller, A.: The evolution of mental models. In E., K., ed.: Advances in Genetic Programming. MIT Press (1994) 199–219