A Hyper-Heuristic Classifier for One Dimensional Bin Packing Problems : Improving Classification Accuracy by Attribute Evolution

Kevin Sim, Emma Hart, Ben Paechter

Institute for Informatics and Digital Innovation, Edinburgh Napier University
Merchiston Campus, Edinburgh, EH10 5DT
{k.sim, e.hart, b.paechter}@napier.ac.uk

Abstract. A hyper-heuristic for the one dimensional bin packing problem is presented that uses an Evolutionary Algorithm (EA) to evolve a set of attributes that characterise a problem instance. The EA evolves divisions of variable quantity and dimension that represent ranges of a bin's capacity and are used to train a k-nearest neighbour algorithm. Once trained the classifier selects a single deterministic heuristic to solve each one of a large set of unseen problem instances. The evolved classifier is shown to achieve results significantly better than are obtained by any of the constituent heuristics when used in isolation.

Keywords: Hyper-heuristics, one dimensional bin packing, classifier systems, attribute evolution

1 Introduction

The one dimensional bin packing problem (BPP) is a well researched NP-hard problem which has been tackled using a diverse range of techniques including mathematically complete procedures[16], deterministic heuristics[11], biologically inspired metaheuristics [8] as well as by the field of hyper-heuristics [15]. The plethora of research and benchmark problem instances available combined with the fact that the problem constitutes an integral part of many other more complex problems makes it an ideal domain for investigating new techniques.

This paper presents a hyper-heuristic which attempts to predict which heuristic, from an available pool, will perform best on a given problem instance. The system incorporates a classification algorithm within an EA in an attempt to generate predictor attributes that improve upon the classification accuracy obtained using predetermined characteristics. The system, once trained using half of 1370 benchmark problem instances, achieves results substantially better than any individual heuristic on the unseen problem instances.

The remainder of this paper is organised as follows. The field of hyper-heuristics and related work are introduced in section 2 with the one dimensional bin packing problem domain, the benchmark problem instances and the deterministic heuristics used in this study covered in section 3. The experimental framework is described in section 4 with the results from those experiments presented in section 5. The paper finishes with section 6 where conclusions are drawn and potential for future research is suggested.

2 Hyper-Heuristics

The term hyper-heuristics (HH) first appeared in relation to combinatorial optimisation (CO) problems in [5] although the term was first coined in [6] 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 [9] 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 genetic programming to create new heuristics from constituent component parts [3,4]. All hyper-heuristics, 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 [2,1].

In [15] Ross et al., proposed a hyper-heuristic approach to bin-packing 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[1]. 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 a deterministic selection of a sequence of heuristics for solving each problem instance.

The approach presented here differs in that it does not use pre-defined categories to describe an instance's state. Using a variable-length evolutionary algorithm a set of categories is evolved that when used in conjunction with a classifier algorithm, map the description of an instance to a suitable simple heuristic. In contrast to [15], problem instances are only categorised once and solved using a single heuristic. The motivation behind this is to determine whether it is possible to find an appropriate method of describing a set of problem instances such that each instance can be mapped to the single heuristic that best solves it. The authors of [15] showed this task to be non-trivial and were unable to find a relationship using a perceptron. Whilst the ranges they used to describe a problem appeared "natural" choices they disregard potential relationships between different item sizes that when combined allow for optimal bin packings.

The system presented here, conceptualised in Figure 1 , uses a heuristic selection strategy to choose which from a set of deterministic constructive heuristics to apply to a problem instance based on knowledge of the problem domain obtained during an off-line training phase. This is achieved using a classification algorithm that attempts to match an unseen problem instance to a procedure for solving it based on the problem instance's characteristics. The characteristics used are the percentages of the items with weights within a number of ranges, expressed as ratios of the bin capacity. The divisions used are not fixed in number or dimension but are evolved by the EA during a training phase.

[1] Described in Section 4, Figure 2.

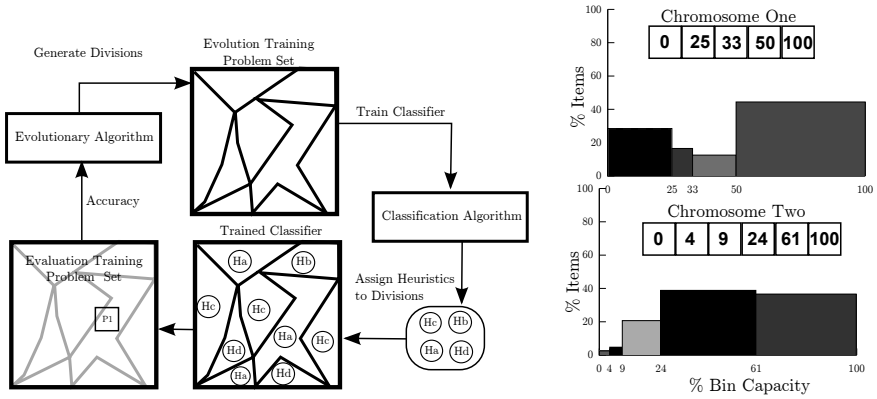


Fig. 1. During off-line training, the EA generates problem divisions, of varying dimension and number, that the classifier assigns the best known heuristic to. The classifier's accuracy in predicting which is the best heuristic for a set of unseen problem instances is used as feedback to the EA. The two graphs show the *same* problem instance encoded by the two different chromosomes shown. The x-axis depicts the evolved ranges expressed as a percentage of the bin capacity whilst the y-axis depicts the percentage of the instances' items with sizes falling within each range.

The system is described in more detail in Section 4 after introducing the BPP domain,benchmark problem instances and heuristics used during this study.

3 One Dimensional Bin Packing Problem

The objective of the one dimensional bin packing problem is to find the optimal number of bins, $OPT(I)$, of fixed capacity c required to accommodate a set of n items, $J = \{\omega_1 \dots \omega_n\}$ with weights $\omega_j : j \in \{1 \dots n\}$ 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 (Scholl, et al., 1997). For any instance $OPT(I)$ 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.

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

Table 1 shows the parameters from which the benchmark data sets used in this study were generated. Data sets $ds1, ds2$ & $ds3$, introduced by Scholl et al., in [16] all have optimal solutions that vary from the lower bound given by Equation 1. However all are known and have been solved since their introduction [17]. All of the instances from $FalU$ and $FalT$, introduced by Falkenauer in [8], have optimal solutions at the lower bound except for one [12].

Four heuristics, three re-created and a fourth introduced here, were included in the system. All pre-sort and select items in decreasing weight order.

Table 1. Data sets $ds1$, $ds3$ and $FalU$ were created by generating n items with weights randomly sampled from a uniform distribution between the bounds given by ω. Those in $FalT$ were generated in a way[8] so that the optimal solution has exactly 3 items in each bin with no free space. Scholl's $ds2$ was created by randomly generating weights from a uniform distribution in the range given by $\varpi \pm \delta$. The final column gives the number of instances generated for each parameter combination.

Data Set	capacity (c)	n	ω	#Problems
$ds1$	100,120,150	50,100,200,500	[1,100],[20,100],[30,100]	$36 \times 20 = 720$
$ds3$	100000	200	[20000,30000]	10
$FalU$	150	120,250,500,1000	[20,100]	$4 \times 20 = 80$
$FalT$	1	60,120,249,501	[0.25,0.5]	$4 \times 20 = 80$

Data Set	c	n	ϖ (avg weight)	$\delta(\%)$	# Problems
$ds2$	1000	50,100,200,500	$\frac{c}{3}, \frac{c}{5}, \frac{c}{7}, \frac{c}{9}$	20,50,90	$48 \times 10 = 480$

- *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* [7] (DJD) and an extension *DJD more Tuples* (DJT) introduced in [15] both pack items into a bin until it is at least a third full. Combinations of up to three (or five for DJT) items are then searched for that best fill the remaining space with preference given to sets that use the largest items. The bin is then closed and the procedure repeats.
- *Adaptive DJD* (ADJD), introduced here, packs items into a bin in descending order until the *free space* in the bin is less than or equal to three times the average size of the items remaining to be packed. It then operates like DJD looking for the set of up to three items that best fills the remaining capacity.

It has been noted [12] that many so called "hard" benchmark problem instances can be solved easily by simple procedures. Often benchmark instances are introduced in the literature alongside procedures specifically designed to solve them, such as those from Falkenauer whose Hybrid Grouping Genetic Algorithm (HGGA) utilises a local search heuristic inspired by Martello and Toth's Reduction Procedure (MTRP) [14] tailored for finding optimal sets of three items. It has been shown for FFD and MTRP[17], and thus DJD and HGGA which both use searches inspired by MTRP, that instances with average weights, $\varpi_j \to \frac{c}{3}$ are the most complex with those where $\varpi_j \to \frac{c}{4}, \frac{c}{5}, \frac{c}{6} \ldots$ proving difficult also.[2] All of the problems used here, except for those in ds2, have an average item weight of around $\frac{c}{3}$.

In [15] the authors showed DJT to be the most successful heuristic when used in isolation solving 73% of instances to the known optimum. The study however omitted $ds2$, on which DJT finds only 45% of the optimal solutions.[3] ADJD, introduced here, whilst the worst performer on the complete set of problems

[2] If a solution exists at the lower bound given in Equation 1 then the total free space $\varpi_{free} \to 0 \; as \; \varpi_j \to \frac{c}{i} : i \in \mathbb{N} : i \geq 3$

[3] DJT will perform best where $\varpi \geq \frac{2}{15}c$ as once the initial filling procedure has filled $\frac{1}{3}c$ the remaining $\frac{2}{3}c$ can be filled by at most five items.

achieves significantly better results on the problem instances from $ds2$. This is accomplished by first packing items in descending order of size until the *free space* in the bin is less than or equal to average size of the items remaining to be packed thus improving the chance of finding a combination of items to fill the remaining capacity for problems with smaller average item weights when compared to DJD or DJT.

In order to get a better indication of a heuristic's performance than can be deduced solely from the number of optimal solutions found, Falkenauer's fitness function, given in Equation 2, is used with k set to 2 in order to reward solutions where any free capacity is restricted to as few bins as possible allowing for a distinction to be made between different solutions that use an equal numbers of bins as well as a measure of a non-optimal solution's quality.

$$f(x) = \sum_{j=1}^{n} \left(\frac{fill_j}{c} \right)^k \div n \tag{2}$$

A third metric used that gives a measure of a heuristic's ability to generalise over a diverse range of problem instances is the number of extra bins required over the optimal number. Table 2 shows the results obtained, for each heuristic, using these three metrics. It is interesting to note for instance, that whilst FFD rates highly if ranked in terms of the number of optimal solutions found, it achieves this using the second largest number of bins. In contrast ADJD, which comes 4^{th} in terms of the number of optimal solutions found, achieves 2^{nd} best position if ranked by either of the other two metrics.

Table 2. The table shows the results obtained by each heuristic on different data sets using three metrics; The percentages of problems solved using the optimum number of bins, the ratio for which the best fitness was attained and the percentage of extra bins required over the optimal. The headings in row 2 depict the data sets as described in Table 1 with Tr and Te depicting the training and test sets used during the experiments described here in section 4 and All representing the complete set of 1370 instances. None of the heuristics used here are able to find optimal solutions to any of the instance from $FalT$ or $ds3$.

Metric	Optimal						Fitness		Bins	
Heuristic	$ds1$	$ds2$	$FalU$	Tr	Te	All	All	Te	All	Te
# Problems	720	480	80	685	685	1370	1370	685	1370	685
FFD	75.83	49.17	7.5	57.66	57.37	57.52	27.52	27.45	1.78	1.81
DJD	79.03	24.05	57.5	52.55	51.97	52.26	47.74	47.74	2.00	2.02
DJT	83.75	44.58	57.5	63.21	62.77	62.99	54.96	55.47	0.73	0.75
ADJD	35.83	80.21	53.75	51.09	49.05	50.07	53.80	52.26	1.12	1.13

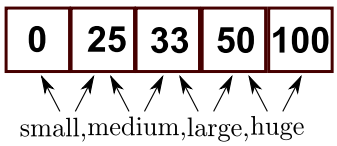
The following section describes the system implemented in an attempt to harness the combined abilities of the heuristics used.

4 Experimental Framework

The system, described in Figure 3, comprises of a database containing the problem instances and corresponding solutions attained by each heuristic along with a classification algorithm and an EA. The classifier predicts which heuristic will perform best on an unseen problem instance whilst the EA attempts to increase classification accuracy by evolving the predictor attributes used. Unlike other applications in which classifiers and EAs have been combined to select which predetermined predictor attributes should be used, the approach here uses the EA to evolve combinations of problem characteristics not known *a priori*. A comprehensive review of EAs combined use with classification algorithms is outwith the scope of this paper for which the reader is directed to [10].

The chromosome representation used by the EA is based on that used in [15] in which an instances' state is described by characteristics which included the percentage of each instances items with weights within certain predetermined ranges, measured as ratios of the bin capacity. The ranges used, and adopted here as a benchmark, are shown in the chromosome representation depicted in Figure 2. These were deemed ''natural" choices by the authors as at most one *Huge*, two *Large* or three *Medium* items can be placed in any individual bin. These ranges, or divisions, are used as the classifiers predictor attributes with the best heuristic being the goal, or class attribute.

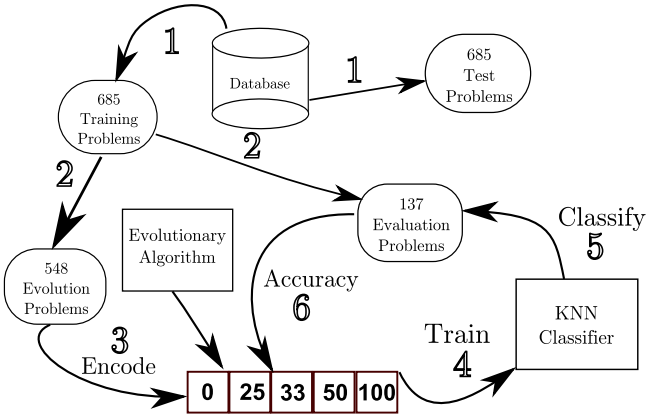
In this study the EA evolves variable length chromosomes which are deliberately constrained to a maximum length that was incrementally increased for each experiment conducted. A chromosome encodes each instance from the *evolution* training set by determining the percentage of items with weights in each range which along with the known best heuristic [4] for each instance is used to train the classifier. The ratio of *evaluation* training problems correctly classified is then used as the objective fitness value. Each data used in this study was created by generating either ten or twenty problems for each parameter combination as described in Table 1. The partitioning of these sets used here ensures an even distribution of instances from each parameter combination between the training and test sets and also the subdivisions of the training set described by Figure 3.



Huge: $\frac{C}{2} < \omega_j$
Large: $\frac{C}{3} < \omega_j \leq \frac{C}{2}$
Medium: $\frac{C}{4} < \omega_j \leq \frac{C}{3}$
Small: $\omega_j < \frac{C}{4}$

Fig. 2. For a chromosome with n genes numbered from left to right the percentage of items p_i falling into each range $r_i < p_i \leq r_{i+1}$ $\forall i = 1, \ldots, n-1$ is encoded and passed to the classifier as predictor attributes. The terminal alleles, 0 & 100 were inferred.

[4] Determined using Equation 2 with ties awarded to the computationally simplest heuristic in the order FFD, DJD, DJT and ADJD.



1. `Separate every alternate problem into training and test sets.`
2. `Split the training set into evolution and evaluation sets with every` 5^{th} `problem put into the evaluation set.`
3. `Using the best chromosome encode the evolution set and use as predictor attributes for the classifier.`
4. `Train the classifier using the predictor attributes with the goal attribute being the best heuristic for each instance.`
5. `Use the classifier to predict the best heuristic for each problem in the evaluation set.`
6. `Measure the classification accuracy and use this figure as the fitness measure for that chromosome.`
7. `After 1000 iterations use the best chromosome and the complete training set to train the classifier.`
8. `The results, presented in section 5, show the ability of the classifier to select the best heuristic for the as yet unseen test set.`

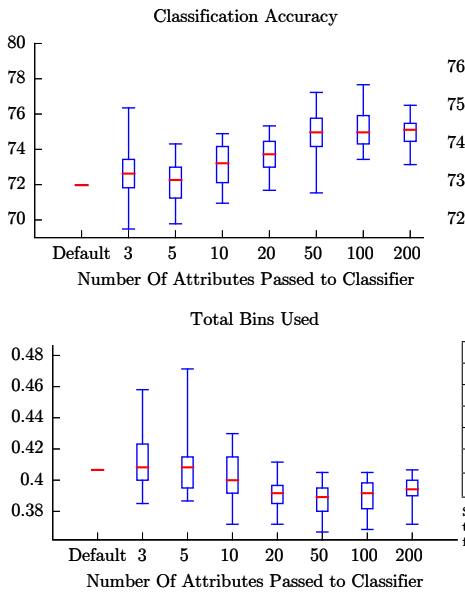
Fig. 3. The system elements and numbered steps explained by the pseudo code

The classification algorithm used was taken from the Waikato Environment for Knowledge Analysis (WEKA) package [13]. After some initial observations a K-Nearest Neighbour Classifier was chosen and used with all parameter settings as default with the exception of the variable k which was set to 2. The EA employed, uses a steady state population, of size 40, with crossover performed to generate one offspring each iteration with a probability of 60%. Each parent is selected by means of a tournament between two randomly chosen competitors. Crossover takes the first parent and selects all alleles up to and including a random position, placing these into the offspring. The second parent is then searched sequentially until an allele value is found greater than has been introduced from the first parent. This and subsequent genes are appended to the offspring. Mutation occurs with a probability of 2% and simply adds or removes, with equal probability, one random value to the chromosome. In order to limit the chromosome length a trimming process is employed. Should the chromosome produced exceed the maximum length stipulated then the closest two allele values are merged, taking on the average value of the two. This trimming procedure

is repeated as necessary until the chromosome is at most the maximum length allowed for that experiment. Each iteration the worst member of the population is replaced by the child if its fitness is better and an identical chromosome does not already exist in the population. Seven experiments were conducted, each consisting of thirty runs with each run terminated after 1000 iterations. For each experiment, the only parameter modified was the maximum allowed chromosome length, l. The values used were $l = \{3, 5, 10, 20, 50, 100, 200\}$. A chromosome length of l corresponds to $l + 1$ ranges once the terminal alleles representing 0 & 100 were added.

5 Results

The results obtained are shown in Figure 4. The best single individual heuristic,



	Accuracy		Solved		Bins	
Att	3	200	3	200	3	200
Mean	72.62	74.93	73.40	74.74	0.41	0.39
SD	1.32	0.83	0.75	0.45	0.018	0.008
Normal	Y	Y	Y	Y	N	Y
t-test	1.27^{-10}		6.84^{-11}			
Wilcoxon					5.86^{-06}	

Shown are the statistical test results obtained for each graph by comparing the data found for 3 and 200 attributes.

Fig. 4. The three plots, taken over 30 runs show, for the unseen 685 test problems, the percentages of problems correctly classified and solved to the known optimal along with the percentage of extra bins over the optimal of 60257 required. The default values show the results obtained when using benchmark attributes (0.25,0.33,0.5). The results of two unpaired two tailed t-tests with no assumption of equal sample variance are given for the data sets that a Shapiro-Wilk Normality test reported as being normally distributed with a non-parametric Wilcoxon Mann-Witney test used for the other.

when ranked by the number of optimal solutions found, was DJT which solved 62.77% (430) of the instances in the test set using an extra 0.75% more bins (452) than the optimum. In comparison the hyper-heuristic presented here found 521

(76.06%) optimal solutions using only 0.37% (223) more bins. A ten fold cross-validation was also conducted using the complete set of 1370 problems and the best set of evolved predictor attributes achieving 72.99% accuracy in comparison to 68.90% using the non-evolved default attributes.

Unlike in [15], the system described here is unable to solve any instances to the optimum that are unsolved by any of the constituent heuristics. As different heuristics, methodologies and problem instances are used a direct comparison is not entirely possible. However for comparison, when trained using the evolved characteristics that gave the best result in terms of the number of optimal solutions obtained along with the truncated training set of problems used in [15] the system presented here was able to find optimal solutions to 172 of the 223 test problems used in [15] as opposed to 166 reported by the papers authors.

6 Conclusions and Future Work

By combining heuristics the number of optimal solutions found is increased substantially over the number found by any individual heuristic. Furthermore by evolving *relevant* predictor attributes for use by the classifier the goal of generating a problem description that maps individual instances to an appropriate heuristic for solving it was achieved. The system developed is able to better generalise over a wider range of problem instances with varying characteristics than can be addressed by any of the heuristics when used in isolation. The new heuristic introduced, ADJD, has been shown to perform better on problem instances with certain characteristics than any of the other heuristics investigated and although the single worst heuristic over the complete set of benchmark instances it is shown to increase the generality of the hyper-heuristic system presented.

It is intended to investigate expanding the work presented here in a number of directions. Separate classifiers, one or more for each heuristic, could be combined with each attempting to predict the fitness that its associated heuristic would achieve when presented with an unseen problem instance. This would allow for multiple classifiers, even of different types, to compete potentially giving rise to improved accuracy in a similar way to ensemble classification techniques.

Another possible direction for further study is to closer emulate the research that inspired this work, where rather than using one heuristic to completely solve a problem instance, a sequence of different heuristics is used which are predicted after each new bin has been packed. All of the heuristics work in this manner already with the exception of FFD which is easily adapted, as in [15], to exhibit the same behaviour. Initial investigations into increasing the number and variety of heuristics used suggests that whilst the classification task increases in complexity with the number of heuristics used, the potential for solving more instances increases also. Although other heuristics were investigated initially many were deemed too similar, such as BFD which found only one optimal solution that FFD did not. The use of Genetic Programming techniques to generate new heuristics could potentially allow for a broader set of simple heuristics with a more diverse range of abilities to be incorporated such as has been investigated in [4] albeit using a considerably smaller set of ninety benchmark instances.

References

1. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: 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)
2. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: Handbook of Metaheuristics, chap. 16, pp. 457–474. International Series in Operations Research & Management Science, Kluwer (2003)
3. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R.: A classification of hyper-heuristic approaches. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 146, pp. 449–468. Springer US (2010)
4. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automating the packing heuristic design process with genetic programming. Evol. Comput. 20(1), 63–89 (Mar 2012)
5. Cowling, P.I., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III, pp. 176–190. PATAT '00, Springer-Verlag, London, UK (2000)
6. Denzinger, J., Fuchs, M.: High performance atp systems by combining several ai methods. In: Proceedings Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97). pp. 102–107. Morgan Kaufmann (1997)
7. Djang, P.A., Finch, P.R.: Solving one dimensional bin packing problems (1998)
8. Falkenauer, E.: A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics 2, 5–30 (1996)
9. Fisher, H., Thompson, G.L.: Probabilistic learning combinations of local job-shop scheduling rules. In: Muth, J., Thompson, G.L. (eds.) Industrial Scheduling, pp. 225–251. Prentice Hall, Englewood Cliffs, New Jersey (1963)
10. Freitas, A.A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer (2002)
11. Garey, M.R., Johnson, D.S.: 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)
12. Gent, I.P.: Heuristic solution of open bin packing problems. Journal of Heuristics 3(4), 299–304 (1998)
13. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explorations 11(1) (2009)
14. Martello, S., Toth, P.: Lower bounds and reduction procedures for the bin packing problem. Discrete Applied Mathematics 28(1), 59–70 (1990)
15. Ross, P., Schulenburg, S., Marín-Blázquez, J.G., Hart, E.: Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 942–948. GECCO '02, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)
16. Scholl, A., Klein, R., Jürgens, C.: Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Comput. Oper. Res. 24(7), 627–645 (1997)
17. Schwerin, P., Wäscher, G.: 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)