Sparse, Continuous Policy Representations for Uniform Online Bin Packing via Regression of Interpolants

John H. Drake[1], Jerry Swan[2], Geoff Neumann[3] and Ender Özcan[4]

[1] Operational Research Group, Queen Mary University of London, Mile End Road, London, E1 4NS, UK
j.drake@qmul.ac.uk
[2] Department of Computer Science, University of York, York, YO10 5GH, UK
jerry.swan@york.ac.uk
[3] Computing Science and Mathematics, University of Stirling, Stirling, FK9 4LA, UK
gkn@cs.stir.ac.uk
[4] School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
exo@cs.nott.ac.uk

Abstract. Online bin packing is a classic optimisation problem, widely tackled by heuristic methods. In addition to human-designed heuristic packing policies (e.g. first- or best- fit), there has been interest over the last decade in the automatic generation of policies. One of the main limitations of some previously-used policy representations is the trade-off between locality and granularity in the associated search space. In this article, we adopt an interpolation-based representation which has the jointly-desirable properties of being sparse and continuous (i.e. exhibits good genotype-to-phenotype locality). In contrast to previous approaches, the policy space is searchable via real-valued optimization methods. Packing policies using five different interpolation methods are comprehensively compared against a range of existing methods from the literature, and it is determined that the proposed method scales to larger instances than those in the literature.

Keywords: Hyper-heuristics, Online Bin Packing, CMA-ES, Heuristic Generation, Sparse Policy Representations, Metaheuristics, Optimisation

1 Introduction

Bin-packing is a well-known NP-hard problem in combinatorial optimization, in which the goal is to pack a set of items into the smallest possible number of fixed-capacity bins [1]. It has been extensively studied in both its online and offline forms. Whereas the sizes of all of the items to be packed are known in

advance in the offline case, online bin packing [2, 3] requires each piece from a 'lengthy' sequence of items to be considered in individually, with no knowledge of the sizes of the following pieces to pack. A packing policy is a heuristic defining how to pack items of different sizes depending on the currently available space in the set of open bins. Here we consider the one dimensional variant of the online bin packing problem, where items have a fixed width and vary in size in only a single dimension [4]. This problem has a wide range of practical applications in industry, e.g. in stock cutting, where a length of fixed width stock material needs to be cut into shorter segments with the minimum waste [5].

Our approach uses the method of *generative hyper heuristics* [6]. These methods seek to generate new heuristics, operating over a search space of heuristics rather than directly over a space of solutions (e.g. [7, 8]). A number of generative hyper-heuristic approaches exist in the online bin-packing literature, with previous work focussing on generating packing policies using different representations. Some previous methods have used Genetic Programming (GP) to represent a packing policy [9, 10], evolving a scoring metric to rank each choice of bin for the current item under consideration. Other work used a matrix representation to define a packing policy [11]. When using a matrix-based representation, each row of the matrix corresponds to a particular item size and each column to a particular remaining bin capacity. Entries for each $(size, capacity)$ combination define the score for packing an item of that size into a bin with that remaining capacity.

These two representations for packing policies suffer from opposing limitations of the search space they present. Typically, GP suffers from a poor genotype-to-phenotype locality, meaning that small changes to a GP program lead to large changes in the solution and the search landscape is correspondingly rugged. Conversely, the use of a matrix representation suffers from being too *dense*: a large number of changes to the representation are required in order to make a significant difference to its phenotypic expression, tending to necessitate a correspondingly large number of evaluations of the objective function.

In this article, we describe an alternative representation of bin packing policies using *interpolants* that we claim does not suffer from defects present in both GP and matrix-based representations. Interpolants are mathematical functions defined by a set of *control points*, with an associated deterministic formula for values between these points. Interpolants are sparsely represented by their control points, and are constructed specifically so that they exhibit good locality. Searching the space of control points, a vector of real-valued parameters, we test five different interpolation methods to define packing polices for the online one-dimensional bin packing problem. We compare our approach to a number of previous approaches from the literature over 12 sets of instances for this problem.

2 Previous approaches for online one-dimensional bin packing

Given a set of n items, with each item j having an associated weight w_j, and a set of n bins with capacity c, Martello and Toth [1] formulated the bin packing problem as follows:

$$\text{Minimise} \quad \sum_{i=1}^{n} y_i \tag{1}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} w_j x_{ij} \leq c y_i, \qquad i \in N = \{1, ..., n\} \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j \in N \tag{3}$$

$$\text{with} \quad y_i \in \{0, 1\}, \qquad i \in N \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad i \in N, j \in N \tag{5}$$

where y_i denotes whether or not bin i has pieces packed in it, x_{ij} denotes whether or not item j has been packed into bin i. The objective function (Expression 1) minimises the total number of bins used, Expression 2 ensures that the fixed capacity of each bin is respected and Expression 3 ensures that each item is only packed once. The online bin packing variant considers the packing of a 'large' number of items which arrive one at a time and a decision regarding which open bin to place each item needs to be made immediately.

Traditionally, online bin packing problems were solved using deterministic heuristics such as Best Fit (BF) and First Fit (FF) [11]. In FF bins are placed into a fixed order and each item is placed into the first bin with sufficient space [12]. The intention is that bins early on in the sequence will be quickly filled and removed from consideration [11]. However, this method relies on an ordering of the bins and this is not possible in the online case. In BF each item is placed into the fullest bin which has room for it. Where ties occur this algorithm operates like FF [12]. Lee and Lee [4] introduced a Harmonic heuristic, which normalises item sizes, and then separates this interval from (0,1] into non-uniform partitions, each representing a certain type and restricting the number of items than can be placed.

A disadvantage of all of these methods is that they assume that the relationship between the preferable choice of bin on one hand, and space/item size on the other, is smooth. A recent study by Özcan and Parkes found that good (optimal) policies could actually be 'spiky' and complex [11, 13]. Recent research in bin packing has tended to focus on metaheuristic strategies capable of automatically devising policies which are more complex than FF or BF and better suited to solving the problem [11, 14, 13].

One metaheuristic often employed is Genetic Programming (GP). An example of a GP solution to the bin packing problem can be found in the work of

Burke et al. [15]. In this work the trees evolved by GP are used to assign a score to each open bin, indicating the desirability of packing the current item into that bin. This technique was able to automatically generate human-developed policies such as FF, as well as a wide range of alternative policies. Further work by Burke et al. [16] showed that the evolved policies were able to scale effectively to instances much larger than those on which they were trained. Burke et al. [9] evolved heuristics for specific sets of bin packing instances and were able to outperform the classic BF heuristic in some cases. Although this method was able to gain some results comparable to BF, crucially, it was not able to consistently outperform it on a regular basis.

Ross et al. developed a hyper heuristic approach using the XCS learning classifier system [17]. Motivated by the fact that traditional metaheuristics such as GAs generate a single heuristic policy that will likely not adapt if the nature of the problem changes, their approach instead evolved a set of rules through which low-level heuristics can be adapted to a changing problem. As more bins were packed, the state of the problem was analysed and matched to appropriate policies using the rule set. This approach performed well on a range of data sets.

Özcan and Parkes [11] used an approach in which policies were represented as two dimensional matrices, with rows corresponding to remaining bin capacity and columns corresponding to item size. The desirability of placing an item of size s into a bin with remaining capacity r, is provided in each matrix at column s and row r. Each item is then packed into the bin with the highest desirability. Matrices were evolved using a Genetic Algorithm. Unlike the previously discussed approaches based on GP, policies evolved using this representation were able to outperform the BF heuristic. This approach was expanded on in a later paper in which each matrix was viewed as a heuristic with a high number of parameters [14]. A heuristic configuration method called the Iterated Racing Algorithm [18] was then used to tune these parameters. Even though the number of parameters was greater than the number usually found in the problems to which iterated racing is applied, it still managed to improve upon human made heuristics such as BF. The original, Genetic Algorithm based approach was still found to be the more successful of the two approaches. In developing these approaches Özcan and Parkes found that the ideal solution was often one which could not easily be expressed through via an arithmetic function. This demonstrates an advantage over GP which is designed to find solutions, expressed through arithmetic functions [11, 13]. Moreover, it was observed that GP mutations often correspond to large moves within the space of policy matrices [19].

3 Learning mechanisms for packing policies

A packing policy can be implemented as a function of the incoming item size s, by assigning an ordinal value to each bin of remaining capacity r (and also to the empty bin). This can either be a bivariate function $p_2(s, r)$ or else, as is the case here, a univariate function $p_1(r - s)$.

As discussed in Section 1, policy representations (e.g. matrix, GP as above) may be characterized as *dense* or *sparse*, according to the minimum granularity of possible changes to the representation. Since individual matrix elements are independent, the dense matrix representation is clearly maximally fine-grained. Additionally, we can characterize representations as *continuous*[1] if small changes to the input produce correspondingly small changes to the output. The matrix representation is therefore both dense and continuous, whereas the GP representation is comparatively more sparse and less continuous. While continuity of representation is clearly desirable, density is not, since many invocations of the objective function are required in order to learn the corresponding policy. This suggests that it may be advantageous to consider alternative representations that are both sparse and continuous. One such possibility are the variety of function interpolation schemes used in numerical analysis. A function interpolator is an (invariably sparse) representation parametrically defined by a set of *control points*. Notably, this includes splines: piecewise polynomial functions which are continuous by construction.

Our approach is therefore to perform a hyper-parameter search over the vector of control points of a univariate function interpolator, which is then used to implement the packing policy. The hyper-heuristic search space is given by $\mathbb{R}^k$, where k is the number of control points. A candidate solution (represented at the hyper-level by a point in $\mathbb{R}^k$) is used to generate a packing policy by using these k values as the y value of the control points (with corresponding x points equally-spaced across the input domain [0, c] of packing policies). Each value along the along the x-axis corresponds to a potential packing (i.e. $r - s$). A packing policy then ranks the desirability of placing the current item into each bin, using the y value defined by the interpolation scheme for the corresponding x value of $r - s$ for that bin. The interpolation schemes considered are:

- **Linear**: Piecewise linear function.
- **Cubic Spline**: Piecewise degree 3 polynomial function, which is continuous and twice differentiable.
- **Divided Difference**: Interpolation via Newton's method of divided differences, expressing the interpolating polynomial as a linear combination of Newton basis polynomials [20].
- **LOESS**: Piecewise polynomial function obtained via locally weighted least squares [21].
- **Neville**: Polynomial function with degree one less than the number of control points which passes exactly through them [22]. The construction uses Newton polynomials via the method of divided differences.

Fig. 1 plots the values from different interpolation schemes. For the control points given, the plot for Neville visibly coincides with divided difference, however, the resulting function values do exhibit small differences. LOESS is parameterized here by a vector of random weights of length equal to the number of control points — if all weights were the same, then LOESS would coincide

[1] The term 'locality' is often used in this context in evolutionary computation.

with cubic. The piecewise description of interpolators also helps to overcome one of the limitations of expressing packing policies through purely arithmetic functions (i.e. lack of conditional statements) [11].

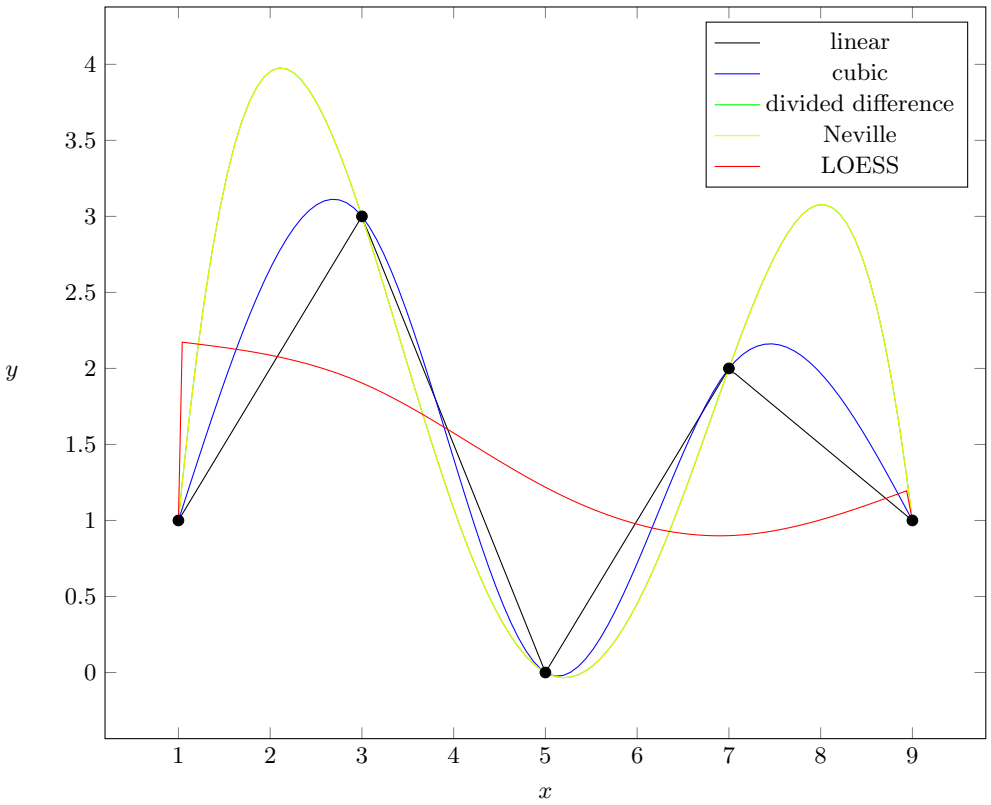

Fig. 1. Interpolations from control points $\{(1,1),(3,3),(5,0),(7,2),(9,1)\}$

As seen in Fig. 1, alternative interpolation schemes define rather different functions. For our purposes, this is not an issue: since we only require good genotype-to-phenotype locality, i.e. from the control points to the corresponding univariate function.

The components used in the hyper-parameter search are as follows:

- **Representation**: For k x-values equally-spaced across the domain of the packing policy function, the solution representation is then a vector in $\mathbb{R}^k$, denoting the corresponding y-values of the k control points.
- **Fitness**: the sum of the *average generic fullness* value, taken over each UBP instance in the training set, where average fullness f for each instance

is calculated as:

$$f = \frac{1}{B} \sum_{t} f_t \tag{6}$$

where B is the number of bins used and f_t is the fullness of bin t.

– **Perturbation**: Solution vector elements are modified according to the mechanism of CMA-ES.

CMA-ES [23] is a well-known and effective metaheuristic for search in $\mathbb{R}^k$. It is one of the most widely-used gradient-free approaches (partly because it requires minimal parameter tuning by the user), and is of particular value when applied to problems with rugged search landscapes. CMA-ES is based upon the foundational Evolutionary Strategies method due to Rechenberg [24], which maintains a companion vector σ of k real values, denoting the mutation step-size to be applied to the corresponding element of the solution vector. CMA-ES further develops the ES approach by adaptively updating mutation step-size via a covariance matrix. Details of the CMA-ES implementation we use in this paper are given in the following section.

4 Experimental framework and results

Our experimental design follows the same methodology used by Asta et al. [13] to enable us to fairly compare against both their technique and the state of the art. Each algorithm compared was tested on a set of progressively larger configurations of the Uniform Bin Packing problem, referred to as *UBP's*. Each UBP is defined by three parameters, maximum bin capacity, minimum item size and maximum item size, denoted as UBP($maxCapacity$,$minItemSize$,$maxItemSize$) herein. An additional parameter, the total number of items, was kept constant at 10^5 in every test. The first 10 UBP problems that we have used were taken directly from the work of Asta et al. [13]. An 11th and 12th have also been introduced in this paper in order to demonstrate the scalability of our technique. These two UBPs, UBP(225,30,150) and UBP(300,40,200), were produced by multiplying each parameter in UBP(150,20,100) by 1.5 and 2 respectively.

For each UBP a two step testing process was used. First of all, a training set consisting of 10 instances of the UBP is randomly generated and candidate packing policies are evolved in one evolutionary run on these training instances. The best packing policy generated is then tested on 100 instances of the UBP, the testing set. Separate training and testing sets ensures that the policies obtained generalise to new problem instances and are not simply obtained by overfitting. Randomization, both of the evolutionary process and of the generation of problem instances, is achieved through the use of the Mersenne Twister random number generator, known to produce a good distribution of random values [25]. In accordance with the recommendations of Luke [26], a set of random seeds is first generated. Each seed is used to generate a separate Mersenne Twister for each UBP. Each interpolant variant is run using the same seeds and tested on the same training and test sets.

As discussed in the previous section, the search over the vector of control points was performed via CMA-ES, a widely-used evolutionary algorithm. The CMA-ES implementation used here was Apache Commons Math, using default parameter settings[2]. After some initial experimentation, the number of control points k was set to 15, with each training run allowed a maximum of 7,500 fitness evaluations. The possible input range for each function is the amount of space left in a bin after adding the current item under consideration, that is $\in [0,(maxCapacity - minItemSize)]$. For each possible bin, the interpolated functions generate a real-valued output score, representing the preference value for that bin.

4.1 Comparison between interpolants and previous results in the literature

The results in Table 1 show the %-average (mean) fullness of the best solution for the interpolant techniques compared to other approaches over 100 instances for each UBP type. Existing methods used for comparison are the classic Best Fit (BF), First Fit (FF), Worst Fit (WF) and Harmonic [4] heuristics as given by Asta et al. [13], in addition to more recent methods: policy matrix based approaches of Yarimcam et al. [14], Asta et al. [13] and Asta and Özcan [27].

From this table, we can see that the performance of different interpolation methods varies depending on the size of the instance considered. In general, we can observe that the interpolant methods offer very good results on the largest instances tested whilst still offering good performance on smaller instances. Linear interpolation is particularly strong in the larger instances, outperforming all other methods on 3 of the 5 largest instance sets. For the largest 3 instance sets we can compare to previous automated policy generation methods: UBP(75,10,50), UBP(80,10,50) and UBP(150,20,100), the best method is always one of the interpolants (twice linear and once divided difference). LOESS and Neville's method of interpolation perform well on the smallest instances, particularly UBP(6,2,3) and UBP(20,5,10) compared to the traditional heuristic methods, however in general they are outperformed by the policy matrix approaches.

Interestingly, the interpolant methods seem to struggle on some mid-size instances when compared to matrix based approaches, particularly in the case of UBP(40,10,20) and UBP(60,15,25). For both of these instance sets, the maximally dense $\text{GA}_{ORIGINAL}$ method outperforms all others. Asta et al. [13] observed that in the smaller instances tested (particularly from UBP(6,2,3) to UBP(15,5,10)), there are disconnected neutralities (plateaus) in the rugged search space of policies for GA to traverse. It could be the case that these mid-sized instances also exhibit this behaviour and are relatively easy for $\text{GA}_{ORIGINAL}$ to traverse. In the case of the interpolant methods, performance might be improved by increasing the number of control points used, creating a denser, more fine-grained space of policies.

[2] https://commons.apache.org/proper/commons-math/javadocs/api-3.6.1/index.html

Table 1. Results obtained using different interpolant-based representations compared to existing methods from the literature in terms of %-average fullness over 100 instances of each UBP type

Methods	UBP(6,2,3)	UBP(15,5,10)	UBP(20,5,10)	UBP(30,4,20)	UBP(30,4,25)	UBP(40,10,20)	UBP(60,15,25)	UBP(75,10,50)	UBP(80,10,50)	UBP(150,20,100)	UBP(225,30,150)	UBP(300,40,200)
Cubic Spline	92.26	94.49	91.64	**99.69**	90.19	96.77	91.90	98.94	93.97	99.04	98.97	98.63
Linear	92.26	95.37	91.28	96.80	98.27	90.21	90.94	**99.33**	96.35	**99.26**	**99.09**	93.37
Divided Difference	92.26	99.49	91.58	99.68	98.39	90.26	92.53	98.80	**99.49**	98.89	98.86	96.67
LOESS	96.56	99.00	97.80	93.16	98.81	90.04	92.28	99.04	98.96	99.09	98.66	**98.97**
Neville	**99.99**	96.56	96.93	99.51	98.14	90.21	92.53	97.07	99.47	98.93	98.75	98.87
BF	92.30	99.62	91.55	96.84	98.38	90.23	92.55	96.08	96.39	95.82	95.73	95.68
FF	92.30	99.55	91.54	96.68	97.93	90.22	92.55	95.91	96.29	95.64	95.55	95.50
WF	91.70	86.58	90.54	88.61	84.10	88.66	90.80	87.94	89.25	87.73	87.64	87.60
Harmonic [4]	-	74.24	90.04	73.82	74.21	89.10	85.18	71.59	72.96	71.97	-	-
$(\pi_r, 10^4)$ [14]	**99.99**	99.60	97.65	99.43	98.46	96.22	99.01	97.17	97.51	-	-	-
$(\pi_r, 10^5)$ [14]	**99.99**	**99.66**	98.21	99.09	99.51	96.81	99.49	96.93	97.36	-	-	-
$GA_{Original}$ [13]	**99.99**	99.63	98.18	99.41	98.39	**96.99**	**99.68**	98.22	98.54	97.88	-	-
GA_{Binary} [13]	**99.99**	99.61	**98.42**	99.58	**99.55**	96.75	96.96	98.45	98.46	97.63	-	-
GA+TA [27]	**99.99**	99.62	98.28	99.53	99.53	96.27	99.47	98.53	98.66	98.22	-	-

It is clear that the policy matrix approaches perform well on the smallest instances, however as the size of the problem increases, the interpolant methods begin to outperform these approaches. As mentioned previously, matrix-based representation is maximally dense, as a desirability score for each (s, r) pair is maintained explicitly. This property restricts this representation in terms of scalability, as an increasingly large number of independent variables must be maintained as the problem size grows. This leads to an incredibly large search space in the case of large instances, which is subsequently much more difficult to search effectively. The search space of policies expressed using interpolant methods is constant irrespective of the problem instance size, as long as the number of control points and the range of values each point can take is fixed.

5 Conclusions

In this paper we have presented a new method of representing packing policies for online bin packing using function interpolation. A policy is defined as a function of the remaining space in a given bin after adding the current item to be packed, providing a score for the desirability of packing the item that bin. Such policies are represented using a set of 'control points', fixed along the input axis, with the exact nature of the function determined by the interpolation method used. Search takes place in hyper-parameter space, across the locations of each control point on the output axis, consisting of a vector of real-valued variables. Unlike previously proposed representations, policies defined using this approach are both sparse and exhibit good locality. Our experiments have shown that policies generated by CMA-ES using this representation can yield better results than both traditional heuristics and state-of-the-art 'policy matrix' approaches, particularly in the case of larger problem instances.

As a result of this work, a number of potential avenues for further research have emerged. One of the limitations of this work is that a fixed number of control points are used. It may be the case that the best choice in terms of number of control points is dependent on the size of the instance being solved, or even differ within a particular instance set depending on the interpolation method used. We intend to explore the relationship between the number of control points used and the number of possible item sizes in an instance and different interpolation methods. Additionally, although here we have chosen to use CMA-ES to search the hyper-parameter space, other continuous optimisation methods such as Genetic Algorithms or Differential Evolution could have been used. Future work will focus on applying other continuous optimisation methods to this problem, assessing their ability to search the hyper-parameter space effectively.

References

1. Martello, S., Toth, P.: Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc. (1990)

2. Csirik, J., Woeginger, G.: On-line packing and covering problems. In Fiat, A., Woeginger, G., eds.: Online Algorithms. Volume 1442 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (1998) 147–177
3. Coffman Jr., E.G., Csirik, J., Galambos, G., Martello, S., Vigo, D.: Bin packing approximation algorithms: Survey and classification. In Pardalos, P.M., Du, D.Z., Graham, R.L., eds.: Handbook of Combinatorial Optimization. Springer New York (2013) 455–531
4. Lee, C.C., Lee, D.T.: A simple on-line bin-packing algorithm. Journal of the ACM **32**(3) (1985) 562–572
5. Sinuany-Stern, Z., Weiner, I.: The one dimensional cutting stock problem using two objectives. Journal of the Operational Research Society **45**(2) (1994) 231–236
6. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R.: A classification of hyper-heuristic approaches. In: Handbook of metaheuristics. Springer (2010) 449–468
7. Woodward, J.R., Swan, J.: The automatic generation of mutation operators for genetic algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2012), ACM (2012) 67–74
8. Drake, J.H., Hyde, M., Ibrahim, K., Ozcan, E.: A genetic programming hyper-heuristic for the multidimensional knapsack problem. Kybernetes **43**(9/10) (2014) 1500–1511
9. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007), ACM (2007) 1559–1565
10. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automating the packing heuristic design process with genetic programming. Evolutionary Computation **20**(1) (2012) 63–89
11. Özcan, E., Parkes, A.J.: Policy matrix evolution for generation of heuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011), ACM (2011) 2011–2018
12. Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L.: Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on Computing **3**(4) (1974) 299–325
13. Asta, S., Özcan, E., Parkes, A.J.: Champ: Creating heuristics via many parameters for online bin packing. Expert Systems with Applications **63** (2016) 208 – 221
14. Yarimcam, A., Asta, S., Özcan, E., Parkes, A.J.: Heuristic generation via parameter tuning for online bin packing. In: IEEE Symposium on Evolving and Autonomous Learning Systems (EALS 2014), IEEE (2014) 102–108
15. Burke, E.K., Hyde, M.R., Kendall, G.: Evolving bin packing heuristics with genetic programming. In: Proceedings of the International Conference on Parallel Problem Solving From Nature (PPSN 2006). Springer (2006) 860–869
16. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.R.: The scalability of evolved on line bin packing heuristics. In: 2007 IEEE Congress on Evolutionary Computation, IEEE (2007) 2530–2537
17. 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 (GECCO 2002). (2002) 942–948
18. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3** (2016) 43–58

19. Parkes, A.J., Özcan, E., Hyde, M.R.: Matrix analysis of genetic programming mutation. In Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C., eds.: Genetic Programming: 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings, Berlin, Heidelberg, Springer Berlin Heidelberg (2012) 158–169
20. Abramowitz, M., Stegun, I.: Handbook of Mathematical Functions. Dover Publications (1965)
21. Cleveland, W.S.: Robust Locally Weighted Regression and Smoothing Scatterplots. Journal of the American Statistical Association **74**(368) (1979) 829–836
22. Stoer, J., Bulirsch, R.: Introduction to numerical analysis. Texts in applied mathematics. Springer (2002)
23. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2) (2001) 159–195
24. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Number 15 in Problemata. Frommann-Holzboog, Stuttgart-Bad Cannstatt (1973)
25. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation (TOMACS) **8**(1) (1998) 3–30
26. Luke, S.: Essentials of metaheuristics. second edn. Lulu (2013)
27. Asta, S., Özcan, E.: A tensor analysis improved genetic algorithm for online bin packing. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, New York, NY, USA, ACM (2015) 799–806