Generalizing Hyper-heuristics via Apprenticeship Learning

Shahriar Asta[1], Ender Özcan[1], Andrew J. Parkes[1], and A. Şima Etaner-Uyar[2]

[1] School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, U.K.
{sba,exo,ajp}@cs.nott.ac.uk
http://cs.nott.ac.uk/~{sba,exo,ajp}/
[2] Department of Computer Engineering
Istanbul Technical University
Istanbul, 34469, Turkey
etaner@itu.edu.tr
http://web.itu.edu.tr/~etaner/

Abstract. An *apprenticeship-learning*-based technique is used as a hyper-heuristic to generate heuristics for an online combinatorial problem. It observes and learns from the actions of a known-expert heuristic on small instances, but has the advantage of producing a general heuristic that works well on other larger instances. Specifically, we generate heuristic policies for online bin packing problem by using expert near-optimal policies produced by a hyper-heuristic on small instances, where learning is fast. The "expert" is a policy matrix that defines an index policy, and the apprenticeship learning is based on observation of the action of the expert policy together with a range of features of the bin being considered, and then applying a k-means classification. We show that the generated policy often performs better than the standard best-fit heuristic even when applied to instances much larger than the training set.

Keywords: Hyper-heuristics, learning by demonstration, apprenticeship learning, generalization

1 Introduction and Related Work

Meta-heuristics have long been used to solve optimization problems using many versions of neighborhood search. However, the efficiency of meta-heuristics depend on the problem domain and the neighborhood operator. Thus, meta- heuristics may have different performances on different problem domains or even on different instances of the same problem. In order to overcome these dependencies, automated search techniques have emerged [8–10], and now are often generically called hyper-heuristics. Hyper-heuristics take the search process one level higher to the space of heuristics. That is, there is a higher level (meta-)heuristic which at each instance of time, chooses some low level and often simpler heuristic

to solve the problem. According to one classification [4], hyper-heuristics, like many machine learning problems, can be divided into three categories depending on the feedback mechanism they employ: *on-line learning*, *off-line learning* and *no learning*. If the hyper-heuristic framework learns while searching, it is an on-line learning hyper-heuristic. On the contrary, an off-line learning hyper-heuristic learns prior to the search phase. When no feedback is acquired from the search space, then the corresponding hyper-heuristic framework is a no learning framework. Hyper-heuristics can also be classified into two groups: *selection* and *generation* hyper-heuristic. The former, selects a heuristic among a set of existing heuristics at each phase of the search. The latter, generates new heuristics from components of the existing low level heuristics. Both selection and generation hyper-heuristics can be further categorized into construction or perturbation heuristics. More on hyper-heuristics can be found in [3, 13, 16]. Selection hyper-heuristics have been well studied, and gave rise to the CHeSC 2011 competition[3]; further details of this can be found in [7, 12] and at the CHeSC website, and of the winning hyper-heuristic by Misir et al. in [11].

However, this paper is about generation rather than selection hyper-heuristics, and so rather than the neighbourhood search of CHeSC, we study the generation of heuristics for an online problem. Specifically, we study the online bin-packing problem and follow the policy matrix methods of Özcan and Parkes [14]. In those methods the goal is to produce an 'index policy', that assigns a score to all potential actions and then selects the highest scoring action. This is done by using direct search by a genetic algorithm. Earlier related work on online bin-packing [17] had proposed a hyper-heuristic approach which learns how to choose a heuristic based on the dynamically changing problem state after placement of each item for bin packing. Subsequent work (e.g. [5, 6]) used genetic programming methods to evolve an arithmetic expression for the scoring function within the policy. Parkes, Özcan and Hyde [15] combined previous studies and presented a method based on policy matrices for analysing the effects of the genetic programming mutation operator in a regular run using online bin packing. The policy matrix methods, and the methods of this paper, differ from that of [17], as it attempts to learn a single heuristic rather than learning how to mix them to construct a solution.

Although the policy matrix approach in [14] was effective at generating heuristics with better performance than the standard ones, it had the drawback of directly only applying to a specific set of values for the bin capacity and range of item sizes. In this paper, we describe a method to take policy matrices learned on small instances and generalise them to apply to different instances, and with the particular aim to apply them to larger instances. We used a form of apprenticeship learning (a.k.a learning by demonstration or imitation learning) [1] for generalizing the demonstrations provided by an expert. Apprenticeship learning has a wide range of applications in control and robotics and is heavily based on Inverse Reinforcement Learning (IRL). Although we do not use IRL

[3] Cross-domain Heuristic Search Challenge:
`http://www.asap.cs.nott.ac.uk/external/chesc2011/`

methods directly in our approach, our study is mainly inspired by them. Our method generates a generalized policy by classifying the actions of some expert policies (heuristics) according to each search state, thus it also can be viewed as a hyper-heuristic. We also note that one intention originally motivating the work of [14] was to produce good policy matrices and then data-mine them to learn good patterns. This work is somewhat different in that it does not learn directly from the policy matrix, but rather by observing the decisions that it lead to.

Our method in use is an off-line classification method, needing to be trained on an available dataset, and so in categorization of hyper-heuristics in [4] it best fits into the category of *off-line learning generation hyper-heuristics*. The study here includes only experimental results on a single problem domain, however, we expect the general methods it will also be applicable to other domains.

2 Policy Matrices for Online Bin Packing

2.1 Online Bin Packing Problem

The bin packing problem is known to be a combinatorial NP-hard problem which deals with packing items of different sizes to bins of fixed capacity. The objective is to minimize the number of bins used. Different variants of the bin packing problem exist, one of which is the *online* bin packing problem. In this variant, we are dealing with partitioning a set of integer values into subsets with the constraint that the sum of integers within a subset does not exceed the capacity [14]. Moreover, as a distinguishing feature, items arrive sequentially and each item has to be assigned to a bin before the next one is disclosed. A decision has to be made dynamically at each step based on partial information regarding which bin should be used for placement. This is in contrast to the off-line bin packing problem where there is a complete information on the number of items and their sizes prior to solving a problem instance.

The bin capacity is a constant integer $C > 1$ and the items can have any size in the range $[1, C]$. An open bin has a remaining capacity which can accommodate at least one item assuming that the sizes of items are known. An empty new bin is always available and it is opened if the size of the current item is bigger than the remaining capacity of all open bins. In such a case, the new bin is opened and the item is placed into this new bin. A bin is closed if its remaining space is smaller than the minimum item size. The uniform bin packing instances are represented by the formalism: $UBP(C, s_{min}, s_{max}, N)$ (adopted from [14]) where C is the bin capacity, s_{min} and s_{max} are minimum and maximum item sizes and N is the number of total items. The item sizes at each step are chosen uniformly and independently random from the range $[s_{min}, s_{max}]$. Also, we have the assumption $s_{min} > 0$ and $s_{max} < C$. The fitness measure for each experiment on N items is computed according to the following equation.

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

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

2.2 Matrix Representation of Policies

As discussed earlier, in our framework we need a set of initial policies which work fine in their own domains (a specific UBP here). Our methodology then utilizes these expert policies to form a generalized model over the problem domain. This generalized model is independent of the underlying policy and a framework which generates expert policies on a given instance is sufficient for the task. Due to its simple implementation, ease of use and high performance, we chose to utilize the work in [14] to generate our expert policies. A description of this method is given below.

Özcan and Parkes [14] proposed a hyper-heuristic method to generate matrix policies to solve instances of online bin packing problem. In their method, policy matrix evolution for generation of heuristics, a policy is represented by a matrix of scores (policy matrix). Each row in this matrix represents the remaining bin capacity (r) prior to the item assignment and each column represents the current item size (s) to be assigned to a bin. The values of each matrix element are either -1 for inactive elements (irrelevant (r, s) pairs which never occur) or W_{rs} which is the score associated with assigning item of size s to a bin of remaining capacity r. The value for W_{rs} is chosen from the range $[w_{min}, w_{max}]$. In our experiments we chose $w_{min} = 1$ and $w_{max} = 2$ for simplicity. The policy matrix is then optimized using an off-line learning GA for a given problem instance (a specific UBP as described in Section.2). Each individual is consisted of the values of the active members of the policy matrix. A generation of these individuals is then generated which goes through selection, recombination, mutation and evaluation. Please note that, since a single policy matrix is a heuristic, then the GA is a hyper-heuristic which searches in the space of heuristics. Further detail on this method can be found on [14]. The experimental results show that this method produces reliable policies which solve a given UBP with a high performance.

3 The Proposed Approach

One of the major contributions of this study is to show that each search state can be seen and described as a feature set with which a generalized model can be constructed. Thus, the feature set is a crucial part of our framework since it affects the performance of our method which benefits from classification algorithms (namely k-means). In order to achieve a desirable performance, the extracted features should be instance independent. That is, they should not be dependent on the absolute values of the item size (s), bin capacity (C) and minimum or maximum item size (s_{min} or s_{max}), but rather to depend on relative sizes. Table 1 shows the list of considered features along with their formal and verbal descriptions. The features in Table 1 are extracted for each open bin on the arrival of each new item. The last two features of the feature vector described

in Table 1, are designed to increase the prediction power of the generalized policy. In other words, regardless of the decision of the expert policy on selecting or rejecting the current bin, we have assumed that the item has been assigned to the bin (if it's size does not exceed the bin's remaining capacity) to see what changes such an assignment makes in the search state. The new remaining capacity of such a hypothetical assignment is noted by the symbol $r' = r - s$ in Table 1.

Table 1. Features of the search state. Note that the UBP instance defines the constants C, s_{min}, and s_{max} whereas the variables are s the current item size, and r the remaining capacity in the bin considered, and r' is simply $r - s$.

feature	description
$(s - s_{min})/(s_{max} - s_{min})$	normalized current item size
r/C	normalized remaining capacity of the current bin
s/C	ratio of item size to bin capacity
s/r	ratio of item size to the current bin's remaining capacity
r'/C	normalized remaining capacity of the current bin after a feasible assignment
$(r'/s_{max}) - s_{min}$	ratio of remaining capacity of the current bin after a feasible assignment to the range of item size

In classical machine learning techniques, each row of features in the dataset determines a certain class label. In this work we chose to use the action which the expert policy prefers for each bin, as the label for each row of features (records). Typically, what is being done by the policy matrices, or any other policy in fact, is to either open a new bin or to choose an open bin and assign the item to that bin. Thus, in our work the label determines if the bin is selected (label 1) or rejected (label 0).

Having determined the necessary features for our method, we can now use our expert policies to extract features and their corresponding labels for each search state. That is, we assume that we are in possession of a set of n expert policies $\{\pi_e^1, ..., \pi_e^n\}$ in one dimensional on-line bin packing problem domain. These expert policies are obtained by the policy generation method discussed in Section.2.2. Each expert policy corresponds to a certain UBP. We run each expert policy once, on it's corresponding UBP for a certain and fixed number of items $N = 10^5$. While running, expert features, ϕ_e^t, are extracted for each state of the search (t). Here, ϕ_e^t is a r dimensional vector of features where r is the number of features representing a search state. At the end of each run for a policy π_e^i we will have a set of demonstrations like:

$$\mathcal{D}_{\pi_e^i} = \{(\phi_e^t, a_t) | \pi_e^i\} \tag{2}$$

where a_t is the action at step t. The demonstration sets for all training policies are then merged together to form a dataset.

$$\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_{\pi_e^i} \tag{3}$$

Having the feature vectors and their associated labels, we employ a k-means clustering algorithm to cluster the feature vectors of each class. The k-means algorithm is a semi-parametric method which uses a mixture of densities to estimate the input sample [2]. The distance metric in use (d) is one minus cosine similarity (Eq.6). The clustering process is designed to generate 8 cluster centroid coordinates, 4 of which labeled as selected bins (feature vectors labeled 1) and the rest as rejected bins (feature vectors labeled 0). The number of clusters for each class has been determined experimentally.

$$\phi_{x_j} = \frac{1}{n_j} \sum_{\phi_e^t \in x_j} \phi_e^t \;\approx\; E(\phi_e^t | x_j \in \mathcal{D}) \;,\; \phi_e^t \in x_j \;,\; \exists i \; s.t \; \phi_e^t \in \pi_e^i \tag{4}$$

Here, x_j is the jth centroid and n_j is the number of samples which belong to the centroid j. For an unseen problem instance (a UBP), at each state of the search, say, on the arrival of each new item, for each open bin, the state features are extracted ($\phi^{t'}$) and the closest matching centroid to the current feature vector in terms of cosine similarity is found. In case the centroid has a label 1 the bin is selected for the item assignment according to a probability. The probability is chosen to be 0.99 and is considered to introduce randomness to the decision making process. Eq.5 illustrates the decision making mechanism of the generalized policy, given a feature vector for a bin and a set of centroids.

$$\pi_g = \{a_{x_j} \in \{0,1\} \mid \underset{j}{\operatorname{argmin}}\, d(\phi_{x_j}, \phi^{t'}) \;,\; \phi_{x_j} \in \mathcal{D}\} \tag{5}$$

Here, π_g is the generalized policy, the subscript x_j indicates the jth centroid obtained by the k-means clustering algorithm, a_{x_j} is the action (label) which is associated to the centroid j and d is the distance metric which is given in Eq.6.

$$d(\phi_{x_j}, \phi^{t'}) = 1 - \frac{\sum_r \phi_{x_j} \cdot \phi^{t'}}{\sqrt{\sum_r {\phi_{x_j}}^2} \cdot \sqrt{\sum_r {\phi^{t'}}^2}} \tag{6}$$

The summations in Eq.6 are over r, the dimension of the feature vector which is not shown as index in the equation in order to reduce the complexity of notations.

4 Experiments

Since we have used the policy matrices generated by the method in [14], a first round of training has been performed to obtain a set of expert policies using the hyper-heuristic in [14]. Then each policy matrix is run on it's corresponding instance to obtain a set of features for search states and form a data set ($\mathcal{D}$ in Eq. 3). However, since the underlying machine which performs the clustering is

an Ubuntu 10.10 with 3GB of RAM, it only can handle small datasets. In order to keep the dataset small enough to be processed by the computer, the actions of the expert on each instance is sampled randomly. That is, not all the states are considered for feature extraction. Instead, a feature vector is extracted for each state according to a uniformly random distribution with a probability of 0.15. The dataset is then clustered which represents the expected feature vector of the expert. For unseen instances of the one dimensional bin packing problem, the feature vector for each open bin is extracted and it is determined if the feature vector of the bin belongs to selected or rejected bins (a choice performed by the expert).

4.1 Experimental Design

In order to obtain expert policies a GA framework as in [14] has been used for which the parameter setting is given in Table 2. Except the values of w_{min} and w_{min}, basically, the entries of Table 2 are the settings which were used in [14] and are given here for convenience. It should be noted that these values were suitable for small instances, but tend not to converge for larger instances. The 'expert policies' obtained can sometimes be significantly sub-optimal, due to the large computational resources needed to learn policies in some cases.

Table 2. GA parameter setting

No. of iterations	200	**pop. size**	$\lceil \frac{C}{2} \rceil$
Selection	Tournament	**Tour size**	2
Crossover	Uniform	**Crossover Probability**	1.0
Mutation	Traditional	**Mutation Rate**	0.1
No. of trials	1000	**No. of Items**	10^5
w_{min}	1	w_{max}	2

The problem instances under consideration are a total of 10 instances. We assume that we have the expert policy corresponding to each instance. That is, we used the GA to obtain an expert policy matrix corresponding to each instance. As a consequence, we know the performance of each expert policy on it's corresponding instance, which is used for comparison in later stages of our experiments. However, in order to train and construct the generalized policy (π_g), we utilize only 3 instances to form the dataset and construct our model. We use the k-means algorithm to construct a generalized model of the choices of expert policies on their corresponding instances. The generalized policy then uses the resulting data set and the model to solve the remaining 7 instances. For feature extraction in the training phase the expert policy is run on it's corresponding instance for a single run which contains 10^5 items. For testing purposes, the generalized policy is tested on each problem instance in the test fold for 100 runs, each including 10^5 items. The instances used to train the π_g and form the dataset are $UBP(15, 5, 10, 10^5), UBP(30, 4, 20, 10^5)$ and $UBP(40, 10, 20, 10^5)$.

4.2 Experimental Results

As mentioned earlier, in our experiments, the expert policy performs a single run on its corresponding instance, resulting in the training feature set. Subsequently, the test instances, are used to test the generalized policy. The results of this experiment is shown in Table 3. In order to have a better understanding of the results, also the results of the best fit (BF) heuristic is given as a lower bound. Please note that the reported results in Table 3 for π_e,π_g and best fit are obtained by running each policy for 100 runs on each problem instance.

Table 3. A comparison between the performances of the expert policy (π_e), generalized policy (π_g) and the best fit heuristic(BF) on various unseen instances. Numbers are average bin fullness percentages.

	Average Performance			Max. Performance			Min. Performance		
Instance	π_e	π_g	BF	π_e	π_g	BF	π_e	π_g	BF
$UBP(20, 5, 10, 10^5)$	98.42	94.32	91.55	98.47	94.41	91.66	94.33	94.21	91.46
$UBP(30, 4, 25, 10^5)$	99.68	97.69	98.38	99.76	97.92	98.49	99.52	97.36	98.30
$UBP(50, 10, 25, 10^5)$	99.20	93.32	93.31	99.31	93.41	93.41	99.08	93.25	93.26
$UBP(60, 15, 25, 10^5)$	99.75	93.83	92.54	99.91	94.80	92.65	99.45	92.91	92.42
$UBP(75, 10, 50, 10^5)$	98.45	98.50	96.08	98.51	98.54	96.13	98.37	98.44	96.04
$UBP(80, 10, 50, 10^5)$	98.86	98.17	96.39	98.91	98.21	96.44	98.74	98.13	96.34
$UBP(150, 20, 100, 10^5)$	97.56	98.32	95.81	97.66	98.37	95.87	97.49	98.26	95.76

```
r\s  1 2 3 4 5 6        1 2 3 4 5 6
1:   . . . . . .        . . . . . .
2:   . 2 . . . .        . 2 . . . .
3:   . 1 2 . . .        . 2 2 . . .
4:   . 2 1 . . .        . 2 1 . . .
5:   . . . . . .        . . . . . .
6:   . 2 2 . . .        . 2 2 . . .
```

Fig. 1. The optimal and generalised matrix policies for $UBP(6, 2, 3, 10^5)$

The generalized policy (π_g) does not generate optimal policies for the test instances, however π_g follows the expert policy (π_e) in terms of performance as summarized in Table 3. Figure 1 illustrates the optimal policy along with a near optimal generalized policy yielding 96.45% mean bin fullness for the instance $UBP(6, 2, 3, 10^5)$ while BF generates a performance significantly worse than π_g with a mean bin fullness of 92.25%. The generalized policy is 1 Hamming-distance away from the known optimal, differing at $W_{3,2}$. In the case of the instance $UBP(30, 4, 25, 10^5)$ BF performs slightly better than the generalized

policy, but in all other instances the generalized policy outperforms the BF. The performance differences are statistically significant for $UBP(20, 5, 10, 10^5)$, $UBP(60, 15, 25, 10^5)$ and the rest of the instances. It is observed that π_g is capable of generalizing the expert policies to larger problem instances. All problem instances in the training phase of π_g, are smaller in terms of bin capacity, minimum and maximum item size as compared to the instances in the test set. Applying the generalized policy to larger unseen problem instances still results with a performance similar to that of the expert policy. This achievement is important since by demonstrating expert actions on simple problem instances, our generalized method was able to perform well on larger instances without undergoing the time consuming cycle of genetic evolution.

5 Conclusion and Future Work

In this study, we have used the idea of apprenticeship learning to construct a generalized model of the problem domain using a set of expert policies derived by a hyper-heuristic. We have described each state of the search by a feature vector and used the feature vector to construct the generalized model. Our experiments show that without a need to re-construct new policies for new instances of the problem domain, our model is able to generalize some existing policies to the problem domain. Our conclusion is that this method can be generalized to a cross-domain level. However, in order to achieve such a level of generality, one has to first determine a common feature set which can be exploited in a domain-independent fashion. Our future work is to have an investigation on automatic feature extraction and selection methods for this purpose. Finally, one could 'complete the loop' and use the generalized policy to generate a policy matrix; which could be used to initialize the GA used in [14]. Such an initialization approach, instead of a randomized initialization scheme, may well be expected to reduce the total number of generations to generate a true expert policy on an unseen problem instance.

References

1. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning. pp. 1–8. ICML '04, ACM, New York, NY, USA (2004)
2. Alpaydin, E.: Introduction to Machine Learning (Adaptive Computation and Machine Learning). The MIT Press (2004)
3. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society (2013), to appear
4. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.: Handbook of Metaheuristics, chap. A Classification of Hyper-heuristic Approaches. International Series in Operations Research & Management Science, Springer (2009)

5. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.R.: The scalability of evolved on line bin packing heuristics. In: Srinivasan, D., Wang, L. (eds.) 2007 IEEE Congress on Evolutionary Computation. pp. 2530–2537. IEEE Computational Intelligence Society, IEEE Press, Singapore (Sep 2007)
6. 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 9th annual conference on Genetic and evolutionary computation. pp. 1559–1565. GECCO '07, ACM, New York, NY, USA (2007)
7. Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vazquez-Rodriguez, J.: Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In: Proceedings of the Multidisciplinary International Scheduling Conference (MISTA09). pp. 790–797 (2009)
8. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000. pp. 176–190. Lecture Notes in Computer Science, Springer, Konstanz, Germany (August 2000)
9. Crowston, W.B., Glover, F., Thompson, G.L., Trawick, J.D.: Probabilistic and parametric learning combinations of local job shop scheduling rules. ONR Research memorandum, GSIA, Carnegie Mellon University, Pittsburgh (117) (1963)
10. Fisher, H., Thompson, G.L.: Probabilistic learning combinations of local job-shop scheduling rules. In: Industrial Scheduling. pp. 225–251. Prentice-Hall (1963)
11. Misir, M., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G.: A new hyper-heuristic as a general problem solver: An implementation in HyFlex. Journal of Scheduling (2012)
12. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E.K.: Hyflex: A benchmark framework for cross-domain heuristic search. In: Hao, J.K., Middendorf, M. (eds.) Evolutionary Computation in Combinatorial Optimization, Lecture Notes in Computer Science, vol. 7245, pp. 136–147. Springer Berlin Heidelberg (2012)
13. Özcan, E., Bilgin, B., Korkmaz, E.: A comprehensive analysis of hyper-heuristics. Intell. Data Anal. 12, 3–23 (January 2008)
14. Özcan, E., Parkes, A.J.: Policy matrix evolution for generation of heuristics. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation. pp. 2011–2018. GECCO '11, ACM, New York, NY, USA (2011)
15. Parkes, A.J., Özcan, E., Hyde, M.R.: Matrix analysis of genetic programming mutation. In: Proceedings of the 15th European conference on Genetic Programming. pp. 158–169. EuroGP'12, Springer-Verlag, Berlin, Heidelberg (2012)
16. Ross, P.: Hyper-heuristics. In: Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, chap. 17, pp. 529–556. Springer (2005)
17. Ross, P., Marín-Blázquez, J., Schulenburg, S., Hart, E.: Learning a procedure that can solve hard bin-packing problems: A new GA-based approach to hyper-heuristics. In: Genetic and Evolutionary Computation GECCO 2003, Lecture Notes in Computer Science, vol. 2724, pp. 215–215. Springer Berlin / Heidelberg (2003)