Heuristic Generation via Parameter Tuning for Online Bin Packing

Ahmet Yarimcam*, Shahriar Asta[†], Ender Özcan[†] and Andrew J. Parkes[†]
ASAP Research Group
School of Computer Science
University of Nottingham
Nottingham NG8 1BB, UK
Email: *itxay2@nottingham.ac.uk, [†]{sba, exo, ajp}@cs.nott.ac.uk

***Abstract*—Online bin packing requires immediate decisions to be made for placing an incoming item one at a time into bins of fixed capacity without causing any overflow. The goal is to maximise the average bin fullness after placement of a long stream of items. A recent work describes an approach for solving this problem based on a 'policy matrix' representation in which each decision option is independently given a value and the highest value option is selected. A policy matrix can also be viewed as a heuristic with many parameters and then the search for a good policy matrix can be treated as a parameter tuning process. In this study, we show that the Irace parameter tuning algorithm produces heuristics which outperform the standard human designed heuristics for various instances of the online bin packing problem.**

I. Introduction

Heuristics have long been used to solve optimization problems, mostly whenever exact methods fail to produce high quality solutions in a reasonable time. However, heuristics may have different performances on different problem domains or even on different instances of the same problem. In order to achieve a higher level of generality, automated search techniques have emerged [1], [2] and [3], and now are often generically referred to as *hyper-heuristics*. Hyper-heuristics take the search process to a level higher to the space of heuristics. That is, there is a higher level (meta-)heuristic which at each step of time, chooses/controls/generates some low level heuristics. 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: online learning, off-line learning and no learning. If the hyper-heuristic framework learns while searching, it is an online learning hyper-heuristic. Conversely, an off-line learning hyper-heuristic learns prior to the search phase. When no feedback is exploited 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 type of hyper-heuristics controls and mixes a set of predefined low level heuristics at each step during the search process, while the latter type generates new heuristics from given components. Both selection and generation hyper-heuristics can be further categorized into construction or perturbation heuristics. More on hyper-heuristics can be found in [5], [6] and [7]. Selection hyper-heuristics have been well studied, giving rise to a challenge ([8], [9]) and the winning online learning hyper-heuristic was provided by Misir et al. [10].

In this paper, we specifically study the online bin-packing problem for which decisions have to be made immediately to place each incoming item into bins of fixed capacity without causing any overflow. Moreover, the policy matrix-based approach of Özcan and Parkes [11] is followed. A policy matrix embeds an "index policy" ([12]) representing a heuristic used for deciding which bin to place an item whenever it arrives. An index policy assigns a score to all potential actions and then selects the highest scoring action. In [11], a good policy matrix for a given problem was evolved using a genetic algorithm. Earlier related work on online bin-packing [13] 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. [14], [15]) employed genetic programming to evolve arithmetic expressions representing scoring functions/policies. Parkes, Özcan and Hyde [16] presented a method based on policy matrices for analysing the behaviour of the mutation operator in genetic programming for online bin packing.

Overall, we propose a generation hyper-heuristic which significantly differs from the previous work in which either a genetic programming or genetic algorithm is used to generate heuristics. This study regards the heuristic generation process as a parameter tuning problem in which a fine-grained representation is used, and in which the number of parameters is often (much) higher than usually assumed in standard parameter tuning processes. We have arbitrarily chosen to use the Iterated Racing algorithm [17] in order to find the most appropriate policy. To the best of our knowledge, approaching heuristic generation from a parameter tuning point of view has not been considered elsewhere.

II. Background

In this section, the online bin packing problem is described followed by a description of the policy matrix representation of heuristics which is the core representation used in this study. The term online bin packing problem refers to the one dimensional bin packing problem, where the bin capacity as well as the item sizes are all scalar values, throughout this paper. Finally, a general overview of parameter tuning approaches is given in which our method of choice (Iterated Racing) is discussed in further detail.

A. Online Bin Packing Problem

The bin packing problem is known to be a combinatorial NP-hard problem [18] which deals with packing items of different sizes to bins of fixed capacity. The objective is to minimize the number of bins used. There is a number of variants of the bin packing problem available in the literature, such as, off-line, two dimensional, three dimensional bin packing and more.

In this paper, online bin packing is studied which is the problem of partitioning a set of integer values into subsets while satisfying a constraint in which the sum of the integer values within the subset does not exceed the capacity [11]. In off-line bin packing full information on the number of items and their sizes is given beforehand, prior to solving the problem. In contrast to this, in online bin packing, each item arrives one at a time and an immediate decision has to be made regarding which open bin will receive that item. In other words, the assignment of each item to a bin is a decision based on incomplete information. The next item(s) are not known by the solver.

The bin capacity is a constant integer $C > 1$ and the item sizes are integer values. While an open bin has a remaining capacity which can accommodate at least one item assuming that the sizes of items are known, a bin is considered to be closed if its remaining space is smaller than the minimum item size. A new empty 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.

The uniform bin packing instances produced by a parametrized stochastic generator are represented by the formalism: $UBP(C, s_{min}, s_{max}, N)$ (adopted from [11]) 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. For a specific problem instance, the size of each item for N items is chosen uniformly and independently at random as an integer value from the range $[s_{min}, s_{max}]$. Also, we assume that $s_{min} > 0$ and $s_{max} \leq C$. For example, $UBP(15, 5, 10, 10^4)$ is a random instance generator and represents a class of problem instances. Each problem instance is a sequence of 10^4 integer values, each representing an item size drawn randomly from $\{5, 6, 7, 8, 9, 10\}$. The probability of drawing the same instance (meaning, exactly the same sequence of item sizes) using a predefined generator of $UBP(C, s_{min}, s_{max}, N)$ is very small as it is controlled by the pseudo-random number generator, and the seed applied to it. Different seeds are used for the training and the tests, hence making them different samples from the same distribution. The objective value for a given solution to an instance is measured in terms of mean bin fullness given N items assuming that the capacity is never exceeded:

$$f = \frac{1}{B} \sum_t f_t \tag{1}$$

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

First fit (FF) and best fit (BF) heuristics are human designed heuristics which can be used for online bin packing. First fit places an incoming item to the first open bin that it fits, while best fit puts an item into the open bin which leaves the minimum remaining space after placement.

B. Matrix Representation of Policies

Özcan and Parkes [11] proposed a genetic algorithm (GA) as a hyper-heuristic method to generate heuristics to solve instances of online bin packing problem. In their study, a policy is represented as a matrix of scores, referred to as policy matrix. A sample policy matrix is shown in Figure 1. 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 (indicated with a dot in Figure 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 this study, $w_{min} = 1$ and $w_{max} = n$ where n is the maximum number of active entries among the columns of the matrix. Given a policy matrix, upon the arrival of each item, a specific column within the matrix which corresponds to the size of incoming item is determined. Within this column, the bin capacity with the highest score is found and the item is then assigned to the bin.

Figure 1 provides an example policy matrix for *solving* (any problem instance generated by) $UBP(15, 5, 10, 10^4)$. The last row in the policy matrix contains the score values of the assignment to the empty bin for various item sizes. The value of n is 7 for this instance from the first column having the largest number of active entries. Assuming that during the packing process, an item with a size of 9 arrives, then the column 9 is considered. That column in the figure contains the scores of all bins associated with all possible remaining sizes. In the example, there are only three possible remaining sizes of 9, 10 and 15 which have been associated with the scores of 4,1 and 2, respectively. Assuming that there are two open bins, one with a remaining size of 9 and the other one being the empty bin with remaining size 15, the item is placed into the former bin. This is because the former bin has a score of 4 which is higher than the other option which has a score of 2.

```
 r\s  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
  1:  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
  2:  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
  3:  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
  4:  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
  5:  .  .  .  .  6  .  .  .  .  .  .  .  .  .  .
  6:  .  .  .  .  3  7  .  .  .  .  .  .  .  .  .
  7:  .  .  .  .  6  3  1  .  .  .  .  .  .  .  .
  8:  .  .  .  .  1  1  1  7  .  .  .  .  .  .  .
  9:  .  .  .  .  1  2  5  3  4  .  .  .  .  .  .
 10:  .  .  .  .  5  7  2  7  1  7  .  .  .  .  .
 11:  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 12:  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 13:  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 14:  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
 15:  .  .  .  .  2  3  5  4  2  3  .  .  .  .  .
```

Fig. 1: An example of a policy matrix for the $UBP(15, 5, 10, 10^4)$ generator of instances

C. Previous Work on Policy Matrices

As well as the original study on policy matrices, there have been a number of relevant studies on this topic, encompassing various techniques with which policy matrices are either generated and/or used.

In the original work [11], the policy matrix was optimised using an off-line learning GA for a given problem instance. Each individual is represented by the values of the active members of a policy matrix. A generation of these individuals is then generated which goes through selection, recombination, mutation and evaluation. Since a single policy matrix is a heuristic, then the GA is acting as a hyper-heuristic which searches in the space of heuristics. The experimental results show that this method produces extremely effective policies for a given UBP instance.

In [16], previous studies on policy matrices were combined to present an approach based on policy matrices for analysing the effect of the mutation operator in Genetic Programming (GP) in a regular run using online bin packing. In a separate study [19], a continuation of the original work in [11], a form of dimensional reduction was considered in which entries in the matrix were grouped into elements taken from one-dimensional vectors. This approach effectively reduced the number of variables from quadratic to linear in the bin capacity. It was shown that evolving the one-dimensional vectors, in a manner similar to the evolution of policy matrices, results in high quality solutions. Apprenticeship Learning (AL) technique was also used in [20] to generate new heuristics based on evolved policy matrices. The AL framework observes and learns the actions of a policy matrix (evolved/optimized by the GA framework in [11]) on instances with a small range of item sizes. These observations are then generalized, resulting in a new heuristic. The generated heuristic is shown to be particularly effective on instances with a large range of item sizes.

D. Parameter Tuning

Numerous algorithms have been designed to tackle optimization problems, and they usually have many parameters whose values (often) influence the performance of the algorithm dramatically. Parameter tuning is the process of finding the best (if possible optimal) setting for the parameters of a given optimization algorithm. Many researchers have been working on the ideas to automatically *tune* a given parametrized algorithm (*target algorithm*) to its best performance. However, the best setting for the parameters of a given algorithm yielding a better performance vary with the problem which is being optimized by the target algorithm [21].

There are various approaches and tools provided for parameter tuning. CALIBRA described in [22] is based on an approach which uses a local search algorithm after a Taguchi fractional experimental design. Parameter Iterated Local Search (ParamILS) was proposed in [23] and used on various algorithms and problem instances (examples are [24], [25] and [26]). Briefly, ParamILS starts by an initial set of parameters defined by the user. This initial set is compared with randomly generated parameter values and the best one is chosen to proceed with. An iterated first improvement search is then conducted in the neighbourhood of the chosen parameter values, producing a new parameter configuration. Then, iteratively, a number of parameter settings are generated randomly in the neighbourhood of this parameter configuration and the iterated first improvement method is applied on each of them, resulting in a new best parameter setting. This process continues until a pre-determined time threshold is reached. Two instantiations of the ParamILS exists: BasicILS and FocusedILS. The former considers a fixed number of training instances while the latter determines the number of training instances adaptively, using dominance criterion.

Iterated racing method (Irace) [17] is an automatic configuration method which is based on Iterated F-Race proposed in [27] and [28]. The basic assumption in this method is that each configurable parameter has a sampling distribution of its own which is independent from other parameters. The distribution is considered to be a normal distribution in case of numerical parameters and discrete in case of categorical ones. In Irace, the search is biased towards sampling distribution(s) with which better parameter configurations are found. The selection of the sampled configurations is based on the racing method. Racing was first proposed in [29] and later adapted for automatic algorithm configuration in [30]. In Irace, at each iteration the set of parameter configurations are applied to the target algorithm. Those configurations which are statistically worse than at least another configuration are discarded. A pseudo-code of the Irace algorithm is given in Algorithm 1.

Similar to ParamILS, Irace has been used in different fields for different purposes, including to tune parameters of various optimization algorithms, such as, Ant Colony Optimization [31] and Particle Swarm Optimization [32], to configure the timetabling scheduling algorithm as in [33], and to generate optimization algorithms based on grammar descriptions for bin packing and permutation flowshop scheduling as in [34]. In this study, we use Irace as a part of a dual-phase approach to directly generate heuristics which are fully parametrized in matrix form for online bin packing. The first phase consists of training the system using Irace followed by a testing phase. During the first phase, a set of initial values for the parameters is obtained for the target algorithm over a set of training instances. Those settings are then used while solving unseen problem instances. The goal is to achieve a level of generality of the parameter settings under which the target algorithm performs well over a wide range of unseen problem instances. Indeed, one can view the parameter tuning as a higher level algorithm (hyper-heuristic) searching the space of parameter values of the target algorithm (generating low level heuristics) when the target algorithm is represented as a set of integer values forming the active entries of a policy matrix for online bin packing. Considering the train-test sessions, Irace hyper-heuristic can be categorized as an offline learning generation hyper-heuristic according to the hyper-heuristic classification given in [4].

III. Proposed Approach

As mentioned earlier, we have employed the Irace automatic algorithm configuration method as a search method which performs the search in the space of heuristics (policy matrices). That is, a policy matrix for a class of bin packing instances is seen as a set of parameters. However, comparing to the applications of widely used algorithm configuration packages (such as Irace or ParamILS), in our approach, the number of parameters is higher than usually considered for parameter tuning. Irace is reportedly used to configure a

Algorithm 1 Pseudocode of Iterated Racing Algorithm [17]

```
Require: $I = \{I_1, I_2, ...\}$
    parameter space,$X$
    objective value,$f$
    tuning budget,$B$
$\theta_1 \sim SampleUniform(X)$
$\theta^{elite} = Race(\theta_1, B_1)$
$j = 2$
while $B_{used} \leq B$ do
    $\theta^{new} = Sample(X, \theta^{elite})$
    $\theta_j = \theta^{new} \bigcup \theta^{elite}$
    $\theta^{elite} = Race(\theta_j, B_j)$
    $j = j + 1$
end while
Output: $\theta^{elite}$
```

TABLE I: Number of configurable parameters for each instance generator.

Instance Generator	No. of parameters
$UBP(6, 2, 3)$	7
$UBP(15, 5, 10)$	27
$UBP(20, 5, 10)$	57
$UBP(30, 4, 20)$	272
$UBP(30, 4, 25)$	297
$UBP(40, 10, 20)$	187
$UBP(60, 15, 25)$	297
$UBP(75, 10, 50)$	1517
$UBP(80, 10, 50)$	1722

number of parameters as high as 207 in [35]. To the best of our knowledge, this is the highest number of parameters considered by the Irace package. The highest number of parameters configured by ParamILS is reported to be around 786 [36] where ParamILS is used to configure the WEKA machine learning library [37]. In contrast, in our study, the number of configurable parameters for instances generated by $UBP(75, 10, 50, 10^5)$ and $UBP(80, 10, 50, 10^5)$ are 1517 and 1722 respectively (Table I), both significantly higher than those of the studies mentioned earlier.

We also employ a dual-phase strategy of train and test. Our approach first trains and generates a policy matrix via parameter tuning using one online bin packing problem instance, produced from a fixed UBP generator (see Section II-A). Afterwards, the (*tuned*) policy matrix obtained in the training phase is tested on 100 unseen problem instances which are produced by the same UBP generator. In other words, 100 runs (trials) are performed for each test case. Figure 2 shows the schematic of the training phase of our proposed approach.

As demonstrated in Figure 2, in the first step, the initial policies are constructed and fed into the configuration cycle. The construction of the initial pool of policy matrices consists of automatically determining the number of active entries of the policy matrix (please refer to Section II-B for the definition of active entries). Later, a uniform random population of policy matrices is generated. At this stage, the policy is vectorized and is merely a vector of values which are then columnized and converted to a policy matrix. Each of these matrices are then applied on the online bin packing training instance for one trial. This is in contrast with the original Irace package where a configuration is applied multiple times ($>= 5$) on the

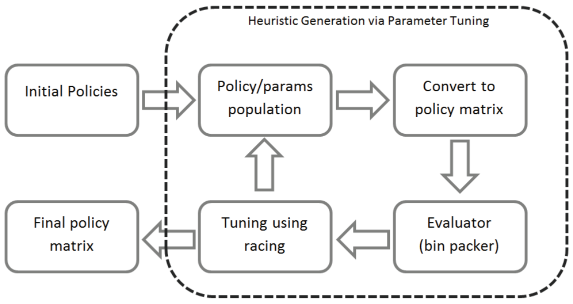


Fig. 2: The proposed approach - training phase

instance whereupon the individuals are ranked statistically. It suits our purpose though to do this without a statistical test as the range of average bin fullness is wide, and even slightest difference is almost all the time significant. Subsequently, the current population of policies are subjected to racing where, again, unlike in original Irace package, a third of the population survives after ranking.

After discarding the worse part of the population, a new population is generated using the surviving individuals. That is, each surviving individual is taken and an offspring is generated by applying a Gaussian distribution ($\mathcal{N}(\mu, \sigma)$) on the elements of the parent. The mean (μ) of the distribution is the current value of the element, whereas the standard deviation (σ) is monotonically decreasing. Centering the normal distribution around the current value of the parent's element and decreasing σ during the tuning process biases the search towards regions close to good performing individuals. The offspring generation is continued until the size of the population reaches that of the previous population. The algorithm then enters a new cycle of evaluation and racing and this process is continued until a predefined number of iterations has been reached.

IV. Experiments

A. Experimental Design

As discussed in Section III, we have employed a dual-phase strategy consisting of a training phase followed by a test session, both of which are considered per generator (per fixed UBP). The training phase is in fact tuning an initially random population of policy matrices to a single policy which performs well for a certain problem instance obtained from the fixed generator. After the training session, testing of a policy is repeated for 100 independent runs, in which, at each run, a different online bin packing problem instance, produced from the same generator, is used. Two classes of train-test sessions are considered in our experiments, depending on the size of the instances that the generator produces. That is, instances generated using the same bin capacity, minimum and maximum item sizes are categorized into *small* ($N = 10^4$) and *large* ($N = 10^5$) instances depending on the number of items. For example, $UBP(6, 2, 3, 10^4)$ generates small instances, while $UBP(6, 2, 3, 10^5)$ generates relatively large ones as the number of items in the latter is 10 times more than the former.

In the first set of train-test sessions, a policy has been trained on a single small instance and tested on a set of unseen large instances. In this set of experiments, the main goal is to see whether any speed-up in the training is possible or not without (unduly) compromising the performance and generality of the tuned policies by training the system on a small instance and applying the generated policy on larger instances. The second set of experiments consists of training using a single large instance and testing the tuned policy on unseen large instances. The training and testing problem instances for each case are of equal size. The former and latter approaches will be referred to as π_{10^4} and π_{10^5}, respectively.

The performance of policies generated by π_{10^4} and π_{10^5} are compared to each other as well as against the Genetic Algorithm (GA) approach as presented in [11], Apprenticeship Learning (AL) in [20] and the Best First (BF) heuristic over the generators in $\{UBP(6,2,3)$, $UBP(15,5,10)$, $UBP(20,5,10)$, $UBP(30,4,20)$, $UBP(30,4,25)$, $UBP(40,10,20)$, $UBP(60,15,25)$, $UBP(75,10,50)$, $UBP(80,10,50)\}$, where N is fixed as 10^5 for testing and so this value is omitted for simplicity from the generators. The BF heuristic simply packs the items in the fullest bin available. The parameter setting of GA is as follows. The GA has a population size equal to half of the bin capacity for a given problem ($\lceil C/2 \rceil$). The selection mechanism is tournament selection with a tour size of 2. A uniform crossover operator with a probability of 1.0 has been chosen for recombination. The traditional mutation operator perturbing a locus with a probability of $\frac{1}{ChromosomeLength}$ is used. A trial is terminated when the maximum iterations of 200 is exceeded. As for the racing algorithm, the same setting as in GA is used for the sake of a fair comparison. The budget in the racing algorithm is set equal to the allowed iteration count in the GA framework. Moreover, since the GA framework also consists of train and test session, all the parameters regarding train and test (population size, number of trials during train and test and etc) are set to the same value as in GA. The tie braking in policy matrices, regardless of the way it was generated (using GA or racing), is First Fit (FF).

B. Experimental Results

Table II provides the best, worst and average results obtained from the policies produced by the algorithms π_{10^4}, π_{10^5}, BF, GA and AL over 100 runs (instances). Moreover, the Wilcoxon signed-rank test is used for evaluation of the average performance difference between various pairs of algorithms and results are summarised in Table III.

Comparing the performance of each policy generated by π_{10^4} and π_{10^5} separately to the BF heuristic shows that on almost all of the cases, the tuned policies outperform the BF heuristic in a statistically significant manner. The only exception is the performance of π_{10^4} on the instances generated by $UBP(15,5,10)$ for which it performs slightly worse than the BF heuristic (see Table III). Even the worst performing tuned policy performs better than the BF heuristic on a given generator for most of the cases. Tuning policy matrices is indeed possible and the automatically generated policies via tuning are capable of outperforming the human designed heuristic for the selected generators in this study.

The average performance comparison between π_{10^4} and π_{10^5} based on % average bin fullness shows that π_{10^5} is better than π_{10^4} in the overall. π_{10^5} wins on 5 generators while π_{10^4} wins on 3 of them, and they both deliver a similar performance on the $UBP(6,2,3)$ generator. The same phenomena is observed considering the best results obtained by each algorithm. Considering the results obtained by each tuning approach for each generator during the experiments, π_{10^4} tends to perform better than π_{10^5} particularly on instances with larger range of item sizes and bin capacity. Hence, training policies via tuning on small instances is a viable strategy and can achieve high quality policies which can be then applied to larger instances yielding a reasonable performance. In some cases, this performance could be even better than training policies on larger instances.

TABLE III: Average performance comparison of various approaches. The Wilcoxon signed rank test is performed on the % average bin fullness over 100 instances (runs) obtained from a pair of algorithms for each UBP generator. Given two algorithms X vs. Y, $\geq$ ($>$) denotes that the algorithm X performs slightly (significantly) better than Y (within a confidence interval of 95%), while $\leq$ ($<$) indicates vice versa. $\approx$ indicates that both algorithms have similar performance. In case there are no reported results for a particular algorithm on a given instance generator, it is indicated by *NA*.

X vs. Y	$UBP(6,2,3)$	$UBP(15,5,10)$	$UBP(20,5,10)$	$UBP(30,4,20)$	$UBP(30,4,25)$	$UBP(40,10,20)$	$UBP(60,15,25)$	$UBP(75,10,50)$	$UBP(80,10,50)$
$(\pi_\mathbf{r}, \mathbf{10^4})$ vs. $(\pi_\mathbf{r}, \mathbf{10^5})$	$\approx$	$<$	$<$	$>$	$<$	$<$	$<$	$>$	$>$
$(\pi_\mathbf{r}, \mathbf{10^4})$ vs. BF	$>$	$\leq$	$>$	$>$	$>$	$>$	$>$	$>$	$>$
$(\pi_\mathbf{r}, \mathbf{10^5})$ vs. BF	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$
$(\pi_\mathbf{r}, \mathbf{10^4})$ vs GA	$\approx$	NA	$<$	NA	NA	$<$	NA	NA	NA
$(\pi_\mathbf{r}, \mathbf{10^5})$ vs GA	$\approx$	NA	$<$	NA	NA	$<$	NA	NA	NA
$(\pi_\mathbf{r}, \mathbf{10^4})$ vs. AL	*NA*	*NA*	$>$	*NA*	$>$	$>$	$>$	$<$	$<$
$(\pi_\mathbf{r}, \mathbf{10^5})$ vs AL	*NA*	*NA*	$>$	*NA*	$>$	$>$	$>$	$<$	$<$

The original GA framework [11] only has reported average results on a couple of instance generators, namely $UBP(6,2,3)$, $UBP(20,5,10)$ and $UBP(40,10,20)$. GA performs significantly better than the tuning approach on the latter two instances. The policy generated via tuning performs very much similar to the one generated by GA for $UBP(6,2,3)$. When the AL approach is compared to the policy matrices tuned via the racing algorithm, we have observed that both proposed approaches outperform AL over four out of six instance generators. On the instances generated by $UBP(75,10,50)$ and $UBP(80,10,50)$, however, the AL approach produces better solutions. All those performance variations between the proposed approaches and AL on each generator are statistically significant. The AL approach is considered to be particularly effective on instances with a large range of item sizes and bin capacity. The results confirm a claim in our previous study [20] that the AL framework tends to perform particularly well on instances of large bin capacity and large range of item sizes. However, on the instances with smaller bin capacity and item sizes, the tuned policies outperform the AL algorithm.

Figure 3 compares the structure of the policy matrix achieved by parameter tuning versus the BF policy. The matrix generated via parameter tuning does not have the smooth structure of human-designed heuristics like BF. This is in line

TABLE II: The performance of policies generated via tuning (π_{10^4} and π_{10^5}), GA and AL approaches over the 100 instances (runs) obtained from each given UBP generator. *Avg* indicates the mean performance while *Min* and *Max* are the best and worst results achieved based on the % average bin fullness. Bold entries correspond to the algorithm which performs the best on average. *NA* indicates that no result is available.

Method		$UBP(6,2,3)$	$UBP(15,5,10)$	$UBP(20,5,10)$	$UBP(30,4,20)$	$UBP(30,4,25)$	$UBP(40,10,20)$	$UBP(60,15,25)$	$UBP(75,10,50)$	$UBP(80,10,50)$
$(\pi_{\mathbf{r}}, \mathbf{10^4})$	**Avg**	**99.99**	99.60	97.65	**99.43**	98.46	96.22	99.01	97.17	97.51
	Min	99.99	99.12	97.51	99.26	98.17	96.38	98.65	96.98	97.45
	Max	100.0	99.90	97.76	99.49	98.73	96.03	99.22	97.38	97.55
$(\pi_{\mathbf{r}}, \mathbf{10^5})$	**Avg**	**99.99**	**99.66**	98.21	99.09	**99.51**	96.81	**99.49**	96.93	97.36
	Min	99.99	99.19	97.87	99.00	99.36	96.65	98.93	96.82	97.27
	Max	100.0	99.91	98.38	99.16	99.61	96.91	99.84	97.05	97.44
BF	**Avg**	92.30	99.62	91.55	96.84	98.38	90.23	96.08	96.39	95.82
	Min	92.26	99.15	91.45	96.80	98.31	90.12	96.04	96.32	95.77
	Max	92.31	99.89	91.62	96.90	98.46	90.31	96.13	96.44	95.87
GA[11]	**Avg**	**99.99**	*NA*	**98.43**	*NA*	*NA*	**97.12**	*NA*	*NA*	*NA*
AL[20]	**Avg**	*NA*	*NA*	94.32	*NA*	97.69	93.83	98.50	**98.17**	**98.32**
	Min	*NA*	*NA*	94.21	*NA*	97.36	92.91	98.44	98.13	98.26
	Max	*NA*	*NA*	94.41	*NA*	97.92	94.80	98.54	98.21	98.37

with the discussion in [11] where it has been demonstrated that heuristics which outperform human-designed methods like BF may have a *spiky* structure.

V. Conclusion

In this study, the Irace parameter tuning method is used to generate effective heuristics represented in terms of policy matrices for online bin packing. A policy matrix is treated as a heuristic with many parameters. Policies generated by the Irace tuning method are observed to be of high quality. These policies not only surpass human-made heuristics in terms of performance, they also deliver better performance on small instances when compared to the policies generated by an apprenticeship learning approach [20]. However, a previously proposed genetic algorithm [11] remains the best method for policy generation. To the best of the authors knowledge, Irace (or any other parameter tuning package) has not been used in the way that it is used in this study to generate heuristics directly. This indicates that optimization via parameter tuning is a viable approach.

References

[1] P. Cowling, G. Kendall, and E. Soubeiga, “A hyperheuristic approach to scheduling a sales summit,” in *Practice and Theory of Automated Timetabling III*, ser. Lecture Notes in Computer Science, E. Burke and W. Erben, Eds. Springer Berlin Heidelberg, 2001, vol. 2079, pp. 176–190.

[2] W. Crowston, *Probabilistic and Parametric Learning Combinations of Local Job Shop Scheduling Rules*, ser. Research memorandum. Defense Technical Information Center, 1963.

[3] H. Fisher and G. L. Thompson, “Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules,” pp. 225–251, 1963.

[4] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, “A classification of hyper-heuristic approaches,” in *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science, M. Gendreau and J.-Y. Potvin, Eds. Springer US, 2010, vol. 146, pp. 449–468.

[5] E. K. Burke, M. Gendreau, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, “Hyper-heuristics: a survey of the state of the art,” *JORS*, vol. 64, no. 12, pp. 1695–1724, 2013.

[6] O. Ülker, E. E. Korkmaz, and E. Özcan, “A grouping genetic algorithm using linear linkage encoding for bin packing,” in *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature: PPSN X*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1140–1149.

[7] P. Ross, “Hyper-heuristics,” in *Search Methodologies*, E. Burke and G. Kendall, Eds. Springer US, 2005, pp. 529–556.

[8] E. K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, and J. A. Vazquez-Rodriguez, “HyFlex: A Flexible Framework for the Design and Analysis of Hyper-heuristics,” in *Multidisciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland*, Dublin, Ireland, 2009, pp. 790–797. [Online]. Available: http://www.asap.cs.nott.ac.uk/publications/pdf/MISTA09HyFlex.pdf

[9] G. Ochoa, M. Hyde, T. Curtois, J. A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, A. J. Parkes, S. Petrovic, and E. K. Burke, “Hyflex: A benchmark framework for cross-domain heuristic search,” in *Proceedings of the 12th European Conference on Evolutionary Computation in Combinatorial Optimization*, ser. EvoCOP’12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 136–147.

[10] M. Mısır, K. Verbeeck, P. D. Causmaecker, and G. V. Berghe, “A new hyper-heuristic as a general problem solver: An implementation in hyflex,” *J. of Scheduling*, vol. 16, no. 3, pp. 291–311, Jun. 2013.

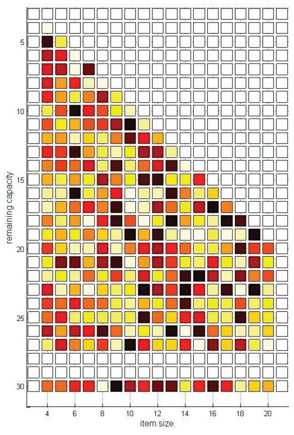
[11] E. Özcan and A. J. Parkes, “Policy matrix evolution for generation of heuristics,” in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO ’11. New York, NY, USA: ACM, 2011, pp. 2011–2018.

[12] Gittins and J. Gittins, “Bandit processes and dynamic allocation,” vol. 41(2), pp. 148–177, 1979.

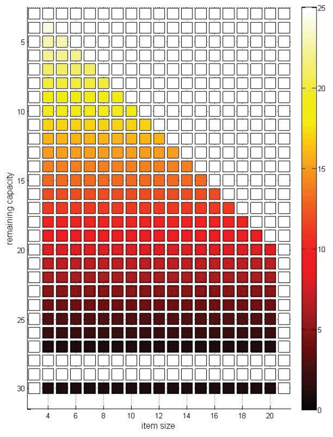
[13] P. Ross, J. G. Marín-Blázquez, S. Schulenburg, and E. Hart, “Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics,” in *Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation: PartII*, ser. GECCO’03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 1295–1306.

[14] E. K. Burke, M. R. Hyde, G. Kendall, and J. R. Woodward, “The scalability of evolved on line bin packing heuristics,” in *2007 IEEE Congress on Evolutionary Computation*, D. Srinivasan and L. Wang, Eds., IEEE Computational Intelligence Society. Singapore: IEEE Press, 25-28 Sep. 2007, pp. 2530–2537.

[15] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, “Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one,” in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO ’07. New York, NY, USA: ACM, 2007, pp. 1559–1565.



(a)



(b)

Fig. 3: Illustration of the (a) best policy matrix obtained via parameter tuning and (b) policy matrix representing BF for $UBP(30, 4, 25, 10^5)$.

[16] A. J. Parkes, E. Özcan, and M. R. Hyde, "Matrix analysis of genetic programming mutation," in *Proceedings of the 15th European Conference on Genetic Programming*, ser. EuroGP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 158–169.

[17] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari, "The irace package, iterated race for automatic algorithm configuration," IRIDIA, Université Libre de Bruxelles, Belgium, Tech. Rep. TR/IRIDIA/2011-004, 2011. [Online]. Available: http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf

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

[19] S. Asta, E. Özcan, and A. J. Parkes, "Dimension reduction in the search for online bin packing policies," in *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, ser. GECCO '13 Companion. New York, NY, USA: ACM, 2013, pp. 65–66.

[20] S. Asta, E. Özcan, A. Parkes, and A. Etaner-Uyar, "Generalizing hyper-heuristics via apprenticeship learning," in *Evolutionary Computation in Combinatorial Optimization*, ser. Lecture Notes in Computer Science, M. Middendorf and C. Blum, Eds. Springer Berlin Heidelberg, 2013, vol. 7832, pp. 169–178.

[21] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[22] B. Adenso-Diaz and M. Laguna, "Fine-tuning of algorithms using fractional experimental designs and local search," *Oper. Res.*, vol. 54, no. 1, pp. 99–114, Jan. 2006.

[23] F. Hutter, D. Babic, H. H. Hoos, and A. J. Hu, "Boosting verification by automatic tuning of decision procedures," in *Proceedings of the Formal Methods in Computer Aided Design*, ser. FMCAD '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 27–34.

[24] A. KhudaBukhsh, L. Xu, H. Hoos, and K. Leyton-Brown, "SATenstein: Automatically Building Local Search SAT Solvers From Components," in *International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 2009.

[25] C. Fawcett, H. H. Hoos, and M. Chiarandini, "An automatically configured modular algorithm for post enrollment course timetabling," University of British Columbia, Department of Computer Science, Tech. Rep., 2009.

[26] C. Thachuk, A. Shmygelska, and H. Hoos, "A replica exchange monte carlo algorithm for protein folding in the hp model," *BMC Bioinformatics*, vol. 8, no. 1, 2007.

[27] P. Balaprakash, M. Birattari, and T. Stützle, "Improvement strategies for the f-race algorithm: Sampling design and iterative refinement," in *Hybrid Metaheuristics*, ser. Lecture Notes in Computer Science, T. Bartz-Beielstein, M. Blesa Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, Eds. Springer Berlin Heidelberg, 2007, vol. 4771, pp. 108–122.

[28] M. Birattari, Z. Yuan, P. Balaprakash, and T. Sttzle, "F-race and iterated f-race: An overview," in *Experimental Methods for the Analysis of Optimization Algorithms*, T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, Eds. Springer Berlin Heidelberg, 2010, pp. 311–336.

[29] O. Maron and A. W. Moore, "The racing algorithm: Model selection for lazy learners," *Artif. Intell. Rev.*, vol. 11, no. 1-5, pp. 193–225, Feb. 1997.

[30] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp, "A racing algorithm for configuring metaheuristics," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 11–18.

[31] M. López-Ibáñez and T. Stützle, "The automatic design of multi-objective ant colony optimization algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 6, pp. 861–875, Dec 2012.

[32] M. Montes de Oca, D. Aydn, and T. Stützle, "An incremental particle swarm for large-scale continuous optimization problems: an example of tuning-in-the-loop (re)design of optimization algorithms," *Soft Computing*, vol. 15, no. 11, pp. 2233–2255, 2011.

[33] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria, "An effective hybrid algorithm for university course timetabling," *Journal of Scheduling*, vol. 9, no. 5, pp. 403–432, 2006.

[34] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle, "Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools," *Comput. Oper. Res.*, vol. 51, pp. 190–199, Nov. 2014.

[35] M. López-Ibáñez and T. Stützle, "Automatically improving the anytime behaviour of optimisation algorithms," *European Journal of Operational Research*, vol. 235, no. 3, pp. 569 – 582, 2014.

[36] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. New York, NY, USA: ACM, 2013, pp. 847–855.

[37] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009.