# CHAMP: Creating Heuristics via Many Parameters for Online Bin Packing

Shahriar Asta, Ender Özcan, Andrew J. Parkes

*ASAP Research Group, School of Computer Science, University of Nottingham, NG8 1BB, Nottingham, UK*

**Abstract**

The online bin packing problem is a well-known bin packing variant which requires immediate decisions to be made for the placement of a lengthy sequence of arriving items of various sizes one at a time into fixed capacity bins without any overflow. The overall goal is maximising the average bin fullness. We investigate a 'policy matrix' representation which assigns a score for each decision option independently and the option with the highest value is chosen for one dimensional online bin packing. A policy matrix might also be considered as a heuristic with many parameters, where each parameter value is a score. We hence investigate a framework which can be used for creating heuristics via many parameters. The proposed framework combines a Genetic Algorithm optimiser, which searches the space of heuristics in policy matrix form, and an online bin packing simulator, which acts as the evaluation function. The empirical results indicate the success of the proposed approach, providing the best solutions for almost all item sequence generators used during the experiments. We also present a novel fitness landscape analysis on the search space of policies. This study hence gives evidence of the potential for automated discovery by intelligent systems of powerful heuristics for online problems; reducing the need for expensive use of human expertise.

*Keywords:* Genetic algorithms, heuristics, packing, decision support systems,

*Email addresses:* ShahriarAsta@gmail.com (Shahriar Asta), Ender.Ozcan@nottingham.ac.uk (Ender Özcan), Andrew.Parkes@nottingham.ac.uk (Andrew J. Parkes)

learning systems, noisy optimization.

## 1. Introduction

There are many problems in which decisions have to be made without fully knowing their future implications, hence, the use of a heuristic 'dispatch policy' is common in such situations. Heuristics are usually designed by an expert through a trial and error process particularly for solving the problems from a specific domain. Since the time-consuming nature of the overall process, automated generation of heuristics has been of interest to both academics and practitioners (Ross, 2005; Chakhlevitch and Cowling, 2008; Burke et al., 2009).

The previous work of Özcan and Parkes (2011) provided an approach for the automatic production of heuristics. The authors formulated the whole process as a special type of parameter tuning (Smit and Eiben, 2009; Yarimcam et al., 2014) in which the number of parameters is much larger than usually considered. This growth in the number of parameters is due to the use of a matrix encoding as a heuristic which represents various potential decisions. In a way, a 'policy matrix' defines an 'index policy' (e.g., Gittins (1979)) covering the possible observed states. A potential decision is given a score separately of other decisions and the decision with the largest score based on a given state is selected.

The online bin packing problem (Coffman et al., 1997; Csirik and Woeginger, 1998) requires immediate decisions to be made for the packing of a lengthy sequence of arriving items of various sizes one at a time into fixed capacity bins without any overflow. In this work, we particularly study the well-known online bin-packing problem, creating a policy that is based on using a (large) matrix[1] of 'heuristic scores'. An observed state for online bin packing considers the item to be packed and each open bin in which that item can fit gets a score from the policy matrix based on its remaining space and the highest value option (open bin) is chosen for packing. The policy matrix can be viewed as a heuristic with

[1] Here, the term 'matrix' is used as a convenience for a 2-d array; there is no implication of it being used for matrix/linear algebra

many parameters requiring optimization yielding scores which can achieve the highest bin fullness in the overall. We describe a framework which allows the use of an optimiser for 'Creating Heuristics viA Many Parameters' (CHAMP) and online bin packing simulator that can be used as an evaluation function for a given policy on a given problem instance. Packing problem instances are specified in terms of a specified bin capacity and a stochastically generated sequence of item sizes taken from a specified range. For specific instance generators, good policies are found using a Genetic Algorithm (GA) as the optimiser under the CHAMP framework to search the space of matrices, with the matrix-based policies being evaluated directly by packing a (large) number of items. The proposed approach can also be considered as an intelligent expert system which captures the decision-making ability of a human expert as a policy matrix and uses a GA to automatically create packing heuristics for online bin packing with the goal of producing better performing packing heuristics than the human designed ones.

The primary result (reported in Özcan and Parkes (2011) and Asta et al. (2013a)) is that the GA finds matrices for the specific packing problems that perform significantly better than the standard general-purpose heuristics such as first and best fit (Rhee and Talagrand, 1993; Coffman Jr et al., 1999). Yarimcam et al. (2014) showed that applying a parameter tuning approach does not match the performance of GA in the overall. Asta et al. (2013b) showed that k-means clustering can be used to generalise the behaviour of GA for solving a given online bin packing problem. Still, GA performed the best. Asta and Özcan (2015) embedded a tensor based learning approach into GA to adapt the mutation rate for each locus. In this paper, we firstly clarify some potentially confusing and counter-intuitive issues in our search methodology and so our experimental design which is different than the previous studies. We then investigate various parameter settings for the proposed policy matrices, including integer and binary settings, as well as two different initialisation schemes.

We also conducted, and present here, a *fitness landscape analysis* (Wright, 1932; Tavares et al., 2008) to better understand the performance of the proposed

algorithm in relation with the policy matrix representation. To the best of our knowledge, this is the first study on fitness landscape analysis on the search space of online bin-packing policies. A fitness landscape is obtained by associating a fitness value indicating the quality of a solution with each point in the search space, on which a neighborhood is defined. In a standard landscape analysis it is the direct solution space that is considered; the novelty of the study here is that instead it is in the space of the heuristics that are used to construct the solutions. It is generally not practical to cover all points in the search space, since the number of points often grows exponentially with respect to the size of a problem. Hence, except for the instance generator with 'small' capacity, sampling is used. Naturally, as standard for fitness landscape analysis, the results are based on a particular neighbourhood (which depends mainly on the solution representation), as well as the move operators employed.

We emphasise that we are aiming to optimise the performance of the heuristic when averaged over many sequences; this is quite different from heuristics (for example, 'Harmonic heuristics', Richey (1991)) designed to be approximation algorithms and so optimise the worst performance achieved over all possible sequences. Also, standard methods for policy creation in stochastic processes (Markov chains, reinforcement learning, etc) are likely to be able to generate good policies in some simpler cases (e.g., Gittins (1979)). However, our driving motivation is to form the basis for evolutionary and other relevant search methods to aid in the generation of heuristics and heuristic policies for complex situations (and out of the reach of analytical methods). For example, situations when sequences are finite (though long) rather than the infinite limit case usually considered in stochastic processes theory (Gittins et al., 2011), or complex non-Markovian time-varying distributions, etc. Such complex situations might include practical combinatorial optimisation problems using constructive heuristics, or queuing networks.

It is important to note we will be creating heuristics for specific (stochastic) generators of item sequences. That is, each usage of CHAMP-GA will be with some specific method of generating sequences of items, the fitness of a heuristic

will be with respect to that generation method, and so the resulting automatically generated heuristic (policy matrix) will be targeting such a generator — though of course the generator is stochastic and so each time the generator is used it will give a different sequence of item sizes, and the policy matrix should work well on all the different sequences.

Our aim is precisely to develop methods that can adapt to specific properties of the sets of sequences of items and exploit their properties. Emphasising again, that policies are evolved to learn from and hence work for (sequences produced by) a generator, and *not* for only one specific sequence. Heuristics tailored to specific generators will generally be expected to perform better than ones aiming to be work on all generators. We believe that such creation of tailored or specialised heuristics is practically useful because users are likely to have input problems with particular characteristics and so would like to be able to develop heuristics that are best able to exploit those characteristics, but would also like to be able to do so automatically, without the need to hire an expert in heuristics. This is particularly relevant if the characteristics of their problems should change over time; in which case, a system should be able to cheaply re-tune the heuristics without having to go back to a human expert. That is, a goal is to develop systems for the creation of heuristics that can reliably, and automatically, adapt to many different situations, without the need for external interventions by experts.

The structure of the paper is as follows: Section 2 gives a brief review and pointers into the literature of existing work on firstly bin-packing and also computer-based methods to help design heuristics. Section 3.1 gives basic definitions of the bin-packing problem, the instance generators that we use, and the existing standard heuristics. Section 3 specifies the policy matrix-based CHAMP framework we use in order to define the packing heuristics. Section 3.4 describes the GA search method we use to find good policies. The experimental methodology is explained in Section 4. Section 5 gives the main results on examples that are large enough to allow policy improvements, but still small enough that the structure of the resulting policies can be (partially) understood. More-

over, a fitness landscape analysis on the search space of policies for online bin packing is provided. Section 6 summarises our results and their implications, and then discusses future plans.

## 2. Related Work

We now briefly cover the bin packing problem and relevant issues, followed by existing methods for automated discovery of heuristics from the scientific literature.

### *2.1. Bin packing*

One dimensional (offline) bin packing is a well known NP-hard (Garey and Johnson, 1990) combinatorial optimisation problem. Solving this problem requires packing of a number of pieces with given sizes into a minimal number of fixed capacity bins. This process can also be considered as partitioning (grouping) a set of integer values into subsets (bins) in such a way that the sum of the integers within a subset does not exceed the fixed capacity. The complete information, including the number of pieces and size of each piece, is known to the solution approaches for the generic one dimensional offline bin packing.

A range of solution approaches from approximation algorithms to metaheuristic optimisation methods has been proposed for bin packing. Metaheuristics provide algorithmic guidelines used for heuristic optimisation (Sörensen and Glover, 2013) and are commonly preferred (Coffman Jr et al., 1999), if the exact approaches fail to produce a solution with reasonable quality within a reasonable time. The representation issue is one of the aspects in bin packing which has been addressed by a number of previous studies, considering that it is a crucial component in metaheuristic design. Falkenauer (1996) introduced a *group encoding* for representing solutions used within a memetic algorithm hybridising a genetic algorithm with local search based on the branch and bound reduction algorithm of Martello and Toth (1990). Ülker et al. (2008) proposed a 'grouping genetic algorithm' framework which utilises a *linear linkage encoding* with the goal of overcoming the redundancies in group encoding. More on bin

packing can be found in Coffman et al. (1997); Csirik and Woeginger (1998). Coffman Jr. et al. (2013) overviews approximation algorithms for bin packing.

The *online bin packing problem* is a variant in which the items are encountered one at a time. In this study, given the size of the piece, we deal with the problem of immediately deciding into which open bin or whether to open a new bin to pack that item at each step before the next item arrives (Lee and Lee, 1985; Richey, 1991). Each decision is made with incomplete information, not knowing what the number of forthcoming pieces and their sizes.

In online one dimensional bin packing, each bin has a capacity $C > 1$ and each item size is a scalar in the range $[1, C]$. More specifically, each item can be chosen from the range $[s_{min}, s_{max}]$ where $s_{min} > 0$ and $s_{max} \leq C$. The items arrive sequentially, meaning that the current item has to be assigned to a bin before the size of the next item is revealed. A new empty bin is always available. That is, if an item is placed in the empty bin, it is referred to as an open bin and a new empty bin is created. Moreover, if the remaining space of an open bin is too small to take in any new item, then the bin is said to be closed.

There are well established heuristics for offline as well as online bin packing among which are First Fit (FF), Best Fit (BF) and Worst Fit (WF) (Johnson et al., 1974; Rhee and Talagrand, 1993; Coffman Jr et al., 1999). The FF heuristic tends to assign items to the first open bin which can afford to take the item. The BF heuristic looks for the bin with the least remaining space to which the current item can be assigned. Finally, WF assigns the item to the bin with the largest remaining space. Harmonic based online bin packing algorithms (Lee and Lee, 1985; Richey, 1991; Seiden, 2002) provide a worst-case performance ratio better than the other heuristics. Assuming that the size of an item is a value in (0,1], the Harmonic algorithm partitions the interval (0,1] into non-uniform subintervals and each incoming item is packed into its category depending on its size. Integer valued item sizes can be normalised and converted into a value in (0,1] for the Harmonic algorithm.

*2.2. Hyper-heuristics for bin packing*

Hyper-heuristics are high level search and optimisation methods which explore the space formed by low level heuristics or heuristic components for solving complex problems (Burke et al., 2003; Ross, 2005; Özcan et al., 2008; Chakhlevitch and Cowling, 2008; Burke et al., 2013). Burke et al. (2010) classified hyper-heuristics into two main categories; methodologies to *select* or *generate* heuristics. This study presents an approach of the latter class used for automated design of heuristics. Here, we cover some relevant work from the scientific literature.

Ross et al. (2002) investigated a learning classifier system, namely Michigan type XCS as a hyper-heuristic generating a hyper-heuristic for offline bin packing. The system is trained across a set of instances to automatically discover the best rule (as a hyper-heuristic) which is capable of identifying the appropriate packing heuristic to use at each decision point based on predefined state features. The XCS approach utilises a reinforcement learning scheme rewarding rules performing well at each step during the search process. Marín-Blázquez and Schulenburg (2006) improved this approach by extending the rewarding mechanism to consider multiple successive steps. In both studies, a large number of instances was split into training and test cases, where the fraction of 'unseen' test instances used was much smaller than the training cases.

Pillay (2012) evaluated the performance of variants of an evolutionary algorithm as hyper-heuristics using a variable length representation encoding a (set of) sequence of low level heuristics for bin packing. The empirical results show that the best performing hyper-heuristic co-evolves two separate circular sequences of heuristics, one to choose an item while the other one to choose the appropriate bin.

The most commonly used generation hyper-heuristic in the scientific literature is Genetic Programming (GP) (Burke et al., 2009) which evolves computer programs based on given components. GP has been applied to many different challenging problems (Poli et al., 2008). Sim and Hart (2013) studied single node genetic programming to generate individual as well as cooperative heuris-

tics for bin packing. The authors evaluated the proposed approach on a large collection of problem instances and report the success of the evolved cooperative heuristics.

A particularly relevant study is provided in Burke et al. (2006). The genetic programming approach is used to evolve heuristics to decide which bin to place a given item. The experimental results showed that GP can produce a heuristic which delivers a performance close to the first fit heuristic. The authors confirmed two issues in their GP approach. Firstly, GP evolved heuristics in tree form yielding very large structures with similar performance, suffering from a well known issue, namely the code-bloat (Bernstein et al., 2004). Secondly, the tree representation yielded redundant solutions. For example, four trees were evolved with depth of 2 delivering similar performances to the first fit heuristic. The follow up work in Burke et al. (2007b) obtained new heuristics generated by genetic programming after a training session and tested those online bin packing heuristics across a collection of randomly generated instances. The authors studied the behaviour of the genetic programming approach as the number of items in the problem instances are varied used in the training against testing phases. The results illustrated that the performance of evolved heuristics improves with the increasing size of training and test problems as expected. The new heuristics discovered by GP delivered a competitive performance to the best fit heuristic. Burke et al. (2007a) extended their study further establishing the compromise between performance and generality level of a generated heuristic for online bin packing.

We also note that a distinct theoretical advantage of the 'policy matrix' approach is that the search space is then at least finite (the number of different matrices is finite though large). In contrast, in the GP approach using functions, the space of functions is infinite – essentially due to the large number of different functions that can generate the same policy matrix and so the same decisions. See Parkes et al. (2012) for a study of the relations of functions.

## 3. Solution Methodology

### *3.1. Online Bin Packing Problem Generator*

The online bin packing instances produced by a parametrised stochastic generator are represented by the formalism: UBP$(C, s_{min}, s_{max}, N)$ (adopted from Özcan and Parkes (2011)) where $C$ is the bin capacity, $s_{min}$ and $s_{max}$ are minimum and maximum item sizes and $N$ is the total number of items. For example, UBP$(15, 5, 10, 10^5)$ is a random instance generator and represents a class of problem instances. A problem instance is a sequence of $10^5$ integer values, each representing an item size drawn independently and uniformly at random from $\{5, 6, 7, 8, 9, 10\}$. The probability of drawing exactly the same instance using a predefined generator of UBP$(C, s_{min}, s_{max}, N)$ is $1/(s_{max} - s_{min} + 1)^N$, producing an extremely low value of $6^{-100000}$ for the example. Note that there are various available instances in the literature (Scholl et al., 1997; Falkenauer, 1996), however, these instances are devised for offline bin packing algorithms and usually consist of a small number of items.

There are two primary ways of utilising random instance generators. A common usage is to create a generator and then generate around a few dozen instances which then become individual public benchmarks. Consequently, methods are tested by giving results on those individual benchmark instance. In our case, the aim is to create heuristics that perform well on average across all instances (where an instance is a long sequence of item sizes) from a given generator. (Hence, for example, we believe it would not serve any useful purpose to place our specific training instance sequences on a website.) An instance generator generally contains a Pseudo-Random Number Generator (PRNG) which needs be supplied with a seed in order to create an instance in the sense of a specific sequence of item sizes.

The choices for UBP represent distributions and not instances of a specific sequence of items; the actual sequence is variable and depends on the seed given to the random number generator used within the item generator. That is, within the instance generator one can use different seed values to generate a different

sequence of items each time the same UBP is used. Indeed, this is the case when we test our approach as it will be seen in the coming sections.

### *3.2. Policy Matrix Representation*

In this study, a packing policy corresponding to a packing heuristic for a given distribution of items (UBP($C, s_{min}, s_{max}, N$)) is defined using a matrix, which will be referred to as a *policy matrix*. We will use the following notation:

- $W$ represents a policy matrix for a given UBP.
- $W_{r,s}$ is an integer score at the $r^{th}$ row and $s^{th}$ column of the policy matrix $W$, where $s_{min} \leq s \leq s_{max}$.
- $W_{r,s} \in [w_{min}, w_{max}]$, that is, $1 \leq w_{min} \leq W_{r,s} \leq w_{max} \leq C$.

In $W$, a score $W_{r,s}$ gives the priority of assigning the current item size $s$ to a remaining bin capacity $r$. Given such a matrix, our approach is to simply scan the remaining capacity of the existing feasible open bins and select the first one to which the highest score is assigned in the matrix (Algorithm1). A feasible open bin is a bin with enough space for the current item size, say $r \geq s$ and includes the always-available new empty bin.

It is clear that the policy matrix is a lower triangular matrix as elements corresponding to $s > r$ do not require a policy (such an assignment is simply not possible). Therefore, only some elements of the policy matrix which correspond to relevant cases for which a handling policy is required are considered. We refer to these elements as active entries while the rest are inactive elements. Inactive entries represent a pair of item size and remaining capacity which either can never occur or are irrelevant.

The active entries along each column of the policy matrix represent a policy with respect to a specific item size and the scores in each column is independent from that of other columns as the policy for a certain item size can be quite different than that of other item sizes.

As an example to clarify how a policy matrix functions, consider that we have a policy matrix for UBP($20, 5, 10, 10^5$) as demonstrated in Figure 1. In

**Algorithm 1:** Applying a policy matrix on a bin packing instance

```
In : W : score matrix;
for each arriving item size s do
    maximumScore = 0;
    for each open bin i in the list with remaining size k do
        if k > s then
            if W_{k,s} > maximumScore then
                maximumScore = W_{k,s};
                maximumIndex = i;
            end
        end
    end
    assign the item to the bin maximumIndex;
    if maximumIndex is the empty bin then
        open a new empty bin and add to the list;
    end
    update the remaining capacity of maximumIndex by subtracting s
    from it;
    if remaining capacity of maximumIndex is none then
        close the bin maximumIndex;
    end
end
```

this matrix, $w_{min} = 1$ and $w_{max} = 12$, which is the maximum number of active items in a column. During the packing process, an item with a size of 5 arrives. The column 5 in the figure corresponds to this item and is used for scoring different options. The column entries represent the scores associated to bins of various remaining capacities. Assume that, currently, two bins with remaining sizes of 9 and 10 are open (the empty bin is always considered to be available). Each one of these bins are associated with scores of 8 and 4 respectively. The score associated to the empty bin is 4. The highest score is thus 8 which corresponds to a remaining bin size of 9. Hence, the incoming item is placed into this bin.

The last row in the policy matrix contains the score values of the assignment to the empty bin for various item sizes. For instance, in the previous example, if the score associated to a remaining bin capacity of 20 (empty bin) was 10 instead of the current value of 4, then a new bin would be opened and the item would be placed in the new (empty) bin.

The tie breaking strategy used in this paper is first fit. (Other tie break policies are possible, but will be explored elsewhere - our main aim here is to demonstrate the methodology and basic results, rather than find the absolute best policy). The first bin is the one whichever has been opened recently. Ties occur when equal scores are associated to bins with different capacities. For example, assume that an item of size 8 arrives. The column corresponding to the new item is the column number 8 which is used for scoring. Also assume that in addition to the empty bin which is always available, bins of remaining capacity of 8, 9 and 10 are open. The scores associated to each of these bins are 2 (for the empty bin), 7, 3 and 7, respectively. In this situation the first bin with the highest score, say, the bin with the remaining capacity of 8 is chosen for item placement. The bin with a remaining capacity of 10 which has an equal score, and is a tie, is ignored along with the empty bin.

| r\s | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|
| 5: | 11 | . | . | . | . | . |
| 6: | 10 | 8 | . | . | . | . |
| 7: | 2 | 3 | 10 | . | . | . |
| 8: | 8 | 5 | 5 | 7 | . | . |
| 9: | 8 | 11 | 1 | 3 | 3 | . |
| 10: | 4 | 1 | 4 | 7 | 5 | 2 |
| 11: | 4 | 9 | 1 | 1 | 8 | 3 |
| 12: | 3 | 4 | 6 | 3 | 1 | 1 |
| 13: | 7 | 11 | 1 | 3 | 4 | 7 |
| 14: | 12 | 10 | 8 | 2 | 7 | 1 |
| 15: | 1 | 2 | 4 | 8 | 5 | 2 |
| 16: | . | . | . | . | . | . |
| 17: | . | . | . | . | . | . |
| 18: | . | . | . | . | . | . |
| 19: | . | . | . | . | . | . |
| 20: | 4 | 9 | 4 | 2 | 2 | 5 |

Figure 1: An example of a policy matrix for UBP$(20, 5, 10, 10^5)$

### *3.3. A Framework for Creating Heuristics via Many Parameters (CHAMP)*

A policy matrix represents a heuristic (scoring function). Changing even a single entry in a policy matrix creates a new heuristic potentially with a different performance. Assuming that each active entry of a policy matrix is a parameter of the heuristic, then a search is required to obtain the best setting for many parameters (in the order of $O(C^2)$).

In this study, we propose a framework for creating heuristics via many parameters (CHAMP) consisting of two main components operating hand in hand: an *optimiser* and a *simulator* as illustrated in Figure 2. CHAMP separates the optimiser that will be creating the heuristics and searching for the best one from the simulator for generality, flexibility and extendibility purposes. The online bin packing simulator acts as an evaluation function and measures how good a given policy is on a given problem.

### *3.4. Evolving Policy Matrices under CHAMP*

Policy matrices are evolved using a Genetic Algorithm (GA) as the optimiser component of the CHAMP framework. Each individual in the GA framework represents the active entries of the score matrix and therefore each gene carries an allele value in $[w_{min}, w_{max}]$. The population of these individuals undergoes the usual cyclic evolutionary process of selection, recombination, mutation and

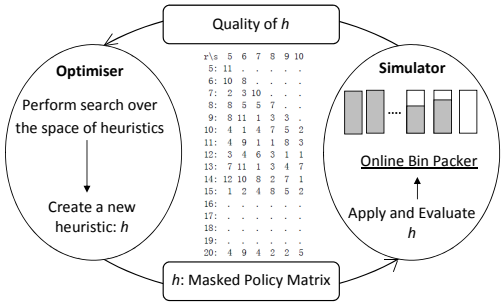



Figure 2: CHAMP framework for the online bin packing problem.

evaluation. Each individual (policy) is evaluated by applying it to the bin packing problem instance as shown in Algorithm 1 using the following fitness function:

> **Average bin fullness,** $F_{af}$: Considering that bin $t$ has a fullness equal to $f_t$, $t \in \{1, \ldots, B\}$ then $F_{af}$ is the value of the occupied space, averaged over the number of used bins. $F_{af} = 1/B \sum_t f_t$, where $B$ is an integer value indicating the number of bins that are used.

At each step, a uniform crossover operator is applied to two parents chosen using the tournament selection method with a tour-size of 2. The traditional mutation operator perturbs two newly generated candidate solutions (offspring) by changing each allele value at a given locus to another value with a probability of $1/L$, where $L$ is the number of active entries in a policy matrix. Then the elitist steady state genetic algorithm replaces two worst individuals of the population with the best two of parents and offspring. This evolutionary process continues for a fixed number of iterations.

The settings for the GA optimiser is given in Table 1.

The GA and the fitness evaluator communicate through the matrices; the GA saves an individual into a matrix and invokes the online bin packing program. The packing algorithm uses the matrix as a policy and evaluates its quality using an instance produced by the instance generator UBP($C$,$s_{min}$,$s_{max}$,$10^5$). The fitness of an individual (policy/heuristic) is computed and then saved into

Table 1: Standard GA parameter settings used during training

| **Parameter** | Value |
|---|---|
| **No. of iterations** | 200 |
| **Pop. size** | $\lceil C/2 \rceil$ |
| **Selection** | Tournament |
| **Tour size** | 2 |
| **Crossover** | Uniform |
| **Crossover Probability** | 1.0 |
| **Mutation** | Random perturbation |
| **Mutation Rate** | $1/L$ |
| **No. of trials** | 1 |

another file for GA to read from. The initial population is randomly generated unless it is mentioned otherwise and the training process continues until a maximum number of iterations is exceeded. A hyper-heuristic operates at a domain-independent level and does not access problem specific information (e.g. see Ross (2005)), thus, the framework we use, as shown in Figure 2, follows the same structure.

In this paper, in contrast to the previous work (Özcan and Parkes, 2011), several instance generators for the one dimensional online bin packing problem have been considered for experiments and $N$ is kept the same during training and testing phases. Three sets of experiments have been performed which includes the original policy matrix evolution scheme and two other variations; one where the encoding is changed to binary and the other where seeding the initial population with first fit heuristic is considered. In Section 5 the results of applying the variants of the framework to problems is reported.

## 4. Experimental Methodology

All our experiments consist of training and test phases. The following subsection provides an overview of how the experiments are set up, measures that are used to evaluate the performance of the algorithms tested and characteristics of the dataset used in the experiments. Section 4.2 illustrates the motivation

behind the unconventional training session with a stochastic algorithm.

*4.1. Experimental Design*

In the training phase, the Genetic Algorithm evolves policies for solving the instances produced from a particular UBP generator. A training session consists of a *single* trial/run. As soon as training is finished, the best policy matrix for the given UBP generator is stored. In the test phase, the bin packer is executed for 101 trials each with a different fixed seed using that best policy matrix. At each trial, a different instance (sequence of item sizes) is produced by the same UBP generator from the training phase. We provide the average bin fullness over 101 trials, achieved by a given algorithm as a measure of its performance on the given UBP generator. Moreover, we have employed a Wilcoxon signed-rank test as a statistical test for pair-wise performance comparison of two given algorithms ($A$ and $B$) based on the fitness values (bin fullness) obtained across 101 trials. The following notation is used. $A \succ / \prec B$ ($A \succeq / \preceq B$) denotes a (slightly) better/worse performance of the left hand side algorithm and that this performance difference is (not) statistically significant within a 99% confidence interval based on the Wilcoxon signed-ranked test. In general, a large number of items, fixed as $N = 10^5$, is produced by the chosen instance generator. However, there are two 'peculiarities', specifically, during each round of training we (generally) use a *single* long item sequence, and with the seeds to the (PRNG in the) instance generator being *fixed*. We clarify these, and their motivations, later in this section as otherwise they might be found rather confusing and counter-intuitive.

During the evolutionary process for training, say, in the main cycle of the GA, when evaluating an individual (a policy matrix) the matrix is given (as input) to the bin packer program — the bin packer is actually the method that provides the fitness function, as it evaluates the quality of the policy matrix by using it to perform packing of a large number of items. The bin packing program hence needs to generate a sequence of items according to the selected range of item sizes, and their probabilities. Naturally, it does this using a pseudo-random

number generator (PRNG) and so needs a 'seed' value to initialise the PRNG.

Hence, for any given UBP, or 'generator instance', the actual sequence if item sizes, or 'sequence instance', needs the capacity, and min/max item sizes, along with a 'seed' for the PRNG. Naturally, we use different seeds for the training and the testing — even though they are working on the same generator, the training and testing will be on different sequences of items. However, different schemes for the selection of the seeds, and so specific item sequences, can also have an effect on the training.

A specific seeding scheme has been employed; we keep a list of seed values stored in a file. When the training starts, in the first generation of the GA, the first seed value is read from the file and is fed to the bin packing program as the same seed value for each element of the population. This is repeated for a fixed number ($m$) generations ($m = 10$ in our experiments). Subsequently, the next number in the list is chosen as the seed value for the upcoming $m$ generations within the GA. This routine continues until the end of the list is reached at which point the seed selection procedure starts over from the beginning of the list. The 'peculiarity' of always using the same seed for each training instance, within the population at each generation, is actually the same as the 'Common Random Numbers' used as a standard technique to reduce variance in simulation and sampling studies. If each element of the population were evaluated using a different seed, that is with a different sequence of item sizes, then the variance between these would swamp the differences between the different policy matrices. We also did explore changing the seed between each generation - so that the training instance(s) would change, more frequently, however, then the search performance became a lot worse. This is worthy of further exploration but we presume it is leading to artificial local minima. Changing the training seed after a small number of generations seemed to form a good compromise (we plan to study this in more detail elsewhere). This scheme also bears some similarity to the well-known method of 'stochastic gradient descent' (SGD). e.g. see Bottou (2012), that is now commonly used in training of artificial neural nets. SGD works by using an improvement method (gradient descent or hill-

climbing) but using a small sample of all the potential data, and then regularly changing the sample. The 'small' sample allows it to work quickly, and the regular change of sample gives a stochastic nature to the search that allows the escape from local minima. In our case the sample is represented by the seed that is used, at each generation of the GA, for the evaluation of the elements of the population, and the regular change of the seed (presumably) allows the population to escape local minima, and so avoid premature convergence to a significantly sub-optimal solution.

In this study, we have used a range of online bin packing instance generators as provided in Table 2. We have performed four sets of experiments. In the first set of experiments, we used a Genetic Algorithm to generate policy matrices for online bin packing with a random initial population and compared the performance of evolved policies and well-known 'human designed' heuristics. In the second set of experiments, we investigated the influence of seeding the initial population with the first fit policy during training phase on the overall approach. In the third set of experiments, we looked into the performance of binary valued policy matrices. Finally, we performed landscape analysis on the online bin packing problems. To the best of our knowledge, this is one of the first studies which performs a fitness landscape analysis on the search space of policies for online bin packing.

Table 2: Problem instance generators of the one dimensional online bin-packing problem which are included in the experiments.

| | |
|---|---|
| UBP$(6, 2, 3, 10^5)$ | UBP$(15, 5, 10, 10^5)$ |
| UBP$(20, 5, 10, 10^5)$ | UBP$(30, 4, 20, 10^5)$ |
| UBP$(30, 4, 25, 10^5)$ | UBP$(40, 10, 20, 10^5)$ |
| UBP$(60, 15, 25, 10^5)$ | UBP$(75, 10, 50, 10^5)$ |
| UBP$(80, 10, 50, 10^5)$ | UBP$(150, 20, 100, 10^5)$ |

*4.2. Single Trial Training and Horizon Effect*

The reason for ever performing a single trial in training (as in Özcan and Parkes (2011)) is from considerations of what we might call a 'horizon effect'. This arises because towards the end (or beginning) of a long sequence of items, the ideal policy might well change from what is the best in the middle of the sequence. Hence, the best policy for a short sequence is likely to be different from that for a long sequence. However, we often want to learn the best policy for an arbitrarily long sequence from a given distribution, and so train on the largest sequence (of fixed length) that is practical in the time allowed. Suppose that we take a concrete example, and have a fixed computational budget for the training phase, with only enough time to pack $10^5$ items. In this case, we need to decide whether the training trial should be a single trial with $10^5$ items, or $R$ trials of $10^5/R$ items each. However, because of the horizon effect, the choice of many short runs is likely to learn a policy that is different from that of a single run. One might think of the many short runs as consisting of a long run but interrupted with 'restarts' in which all bins are closed at regular intervals; however, such restarts are artificial (in the context of trying to learn policies for long runs) and so will distort the learning process. Accordingly, although it does seem counter-intuitive, there is good reason for learning from a single long training sequence as opposed to learning from many (but shorter) ones. Note that a single training run of $10^5$ items does use the matrix $10^5$ times and so is far from a 'single usage'.

To illustrate the influence of using small (finite) and large (infinite) number of items while testing a policy, we have manually generated a "good" policy which enforces that a bin will never become closed unless is perfectly filled for UBP$(9, 1, 9, N)$, as shown in Figure 3. This policy was expected to asymptotically converge to an average bin fullness of 100% ($F_{af}$=1.0) as $N$ grows towards $infinity$ (a large value).

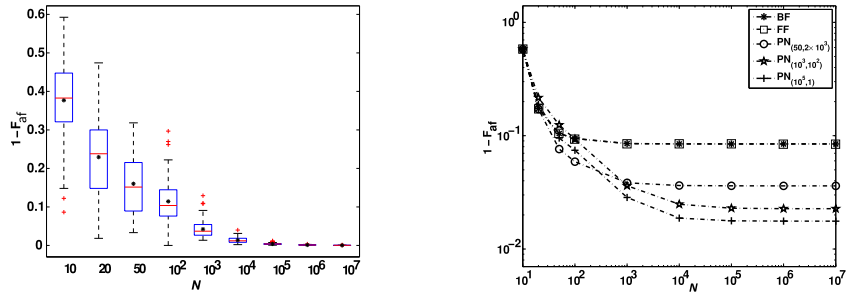
Figure 4(a) clearly shows the *horizon effect* when the policy is tested on 101 instances produced by UBP$(9, 1, 9, N)$ for various $N$ values. As $N$ goes to $10^7$, the expected outcome has indeed been observed and the "good" policy almost

| r\s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1: | 2 | . | . | . | . | . | . | . | . |
| 2: | 1 | 2 | . | . | . | . | . | . | . |
| 3: | 1 | 1 | 2 | . | . | . | . | . | . |
| 4: | 1 | 1 | 1 | 2 | . | . | . | . | . |
| 5: | 1 | 1 | 1 | 1 | 2 | . | . | . | . |
| 6: | 1 | 1 | 1 | 1 | 1 | 2 | . | . | . |
| 7: | 1 | 1 | 1 | 1 | 1 | 1 | 2 | . | . |
| 8: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | . |
| 9: | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

Figure 3: A "good" policy matrix for UBP$(9, 1, 9, N)$

always generates a perfect solution ($F_{af}$=1.0) for any given instance produced by UBP$(9, 1, 9, N)$. Another conclusion we can derive is that the best polices could be totally different for very short sequences. Also, Özcan and Parkes (2011) reported that the performance difference between the worst and best policies obtained from multiple runs on a given instance generator is small; the effect of the choice of the seed to create the single training run does not significantly affect the overall results.

In order to reinforce further why we have chosen to perform a single long training run (large number of items), rather than short multiple runs, we have trained our system executing the bin packer program using $2 \times 10^3$, $10^2$ and 1 randomly generated instance(s), each containing 50, $10^3$ and $10^5$ items, respectively and tested the best policy generated by GA, FF and BF on 101 instances of UBP$(9, 1, 9, N)$ for various $N$. The setting for each short multiple runs ensures that the total number of items used during training is maintained, that is $10^5$. Figure 4(b) illustrates the performance of each heuristic in terms of average (1-$F_{af}$) for each generator UBP$(9, 1, 9, N)$. The plot shows that policy matrices trained and tested on the same number of items result with better or similar average bin fullness when compared to BF/FF. Moreover, BF/FF beats the evolved best policy for $N = 50$. However, an evolved policy regardless of the setting for training beats BF/FF as $N$ goes to 'infinity' (effectively $10^6$ or more). The policy matrix trained by a single run with $10^5$ items yields the best result as $N$ grows towards infinity. The performance of that evolved policy at $N = 10^5$ is close to its $N = 10^6$ or $N = 10^7$ value, which suggests that the




(a) (b)

Figure 4: (a) The performance of a "good" policy matrix on UBP$(9, 1, 9, N)$ for different values of $N$ (number of items). (b) The average performance of BF, FF, and the best policies obtained after training with 50, $10^3$ and $10^5$ items while the bin packer program is executed for $2 \times 10^3$, $10^2$ and 1 time(s), respectively, on UBP$(9, 1, 9, N)$ for various $N$. The best policy obtained with a particular configuration by GA is denoted as 'PN(no. of items,no. of repetitions)'. The $x$-axis uses a log-scale.

training with a single long run is viable.

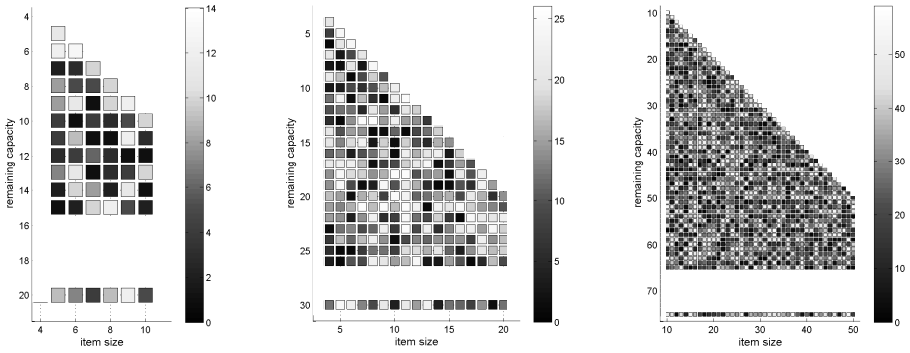
## 5. Results

The analyses of the results obtained from four sets of computational experiments are provided in this section.

### *5.1. Analysis of Genetic Algorithms for Policy Matrix Generation*

This study utilises a non-traditional experimental methodology as explained in Section 4, which is different than the experimental methodology used in Özcan and Parkes (2011). The previous work ignored the horizon effect and applied the best evolved policy found in 50 trials with $N = 500$ items during the training phase. Moreover, the experiments in Özcan and Parkes (2011) were performed on three instance generators. In this study, an extensive analysis of the GA methodology is performed by applying it to a wider range of bin packing instance generators (Table 2). The GA framework operating based on the proposed experimental methodology is referred to as $GA_{Original}$ throughout this

paper. During training, $GA_{Original}$ is used to find a high quality policy matrix for each instance generator (UBP). An example of the best policy matrices of some selected instance generators achieved in the training phase is demonstrated in Figure 5. The percentage average bin fullness (over 101 trials) achieved by each evolved policy matrix for each instance generator during the test phase can be seen in Table 5.




(a) UBP$(20, 5, 10, 10^5)$ (b) UBP$(30, 4, 20, 10^5)$ (c) UBP$(75, 10, 50, 10^5)$

Figure 5: The best policy matrices achieved by $GA_{Original}$ framework for selected problem generators

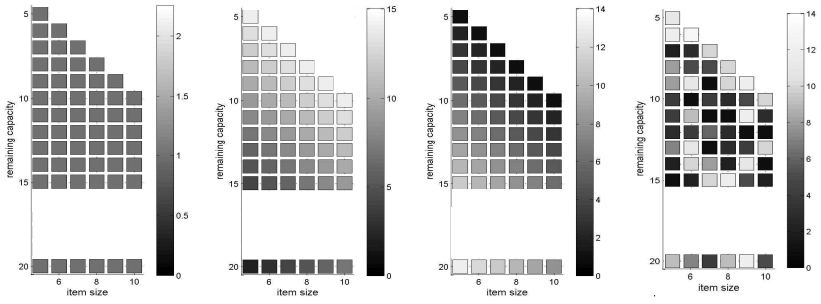
As in Özcan and Parkes (2011), FF, BF and WF heuristics are also applied to the set of problem instances generated by various UBPs. This serves as a baseline for comparison purposes. In this study, for completeness, we have additionally tested the Harmonic algorithm (Lee and Lee, 1985) on the same instance generators. The results show that FF, BF and WF outperform Harmonic (see Table 5) on all instance generators and this performance difference is statistically significant in all cases based on the Wilcoxon signed-rank test. This is understandable as the Harmonic algorithm is designed for the worst case over all sequences, rather than the average case.

FF, BF and WF heuristics can also be represented as policy matrices. Figure 6 shows a policy matrix of each heuristic for the instance generator UBP$(20, 5, 10, 10^5)$. It is easy to note that the policy matrices corresponding to FF, BF and WF

heuristics are smooth matrices. The score values change monotonically across the rows and/or columns in BF and WF matrices whereas the structure of the FF policy matrix is a flat one. Contrary to the smooth structure of simple policy matrices (FF, BF and WF), the structure of the better performing policy matrix discovered by $GA_{Original}$ is "irregular" or even "non-aligned spiky" (rough). Nevertheless, $GA_{Original}$ outperforms FF, BF and WF on all instance classes (all the UBPs). This observation applies to all policy matrices obtained from GA for each problem generator and gives evidence for two main conclusions:

1. There is a need for search-based discovery of the structures within successful policy matrices, since although the rules are easy to derive from a given policy matrix, correlations between item sizes and remaining bin space needs to be detected and this would be hard to create by hand in general.
2. It is unlikely for a 'nice' arithmetic function of remaining capacity and item size to represent those structures.

   This representational issue might well explain why previous work by Burke et al. (2006) was only able to equal the performance of standard heuristics whereas we significantly outperform them.




(a) FF (b) BF (c) WF (d) $GA_{Original}$

Figure 6: Smooth FF, BF and WF policy matrices for solving UBP$(20, 5, 10, 10^5)$ when compared to the spiky structure of the (much better performing) policy matrix achieved by $GA_{Original}$ for the same UBP.

However, we have only demonstrated that some of the best matrices can be 'spiky' ('rough'). We are not claiming here to have shown that all well-performing matrices need to be 'spiky'; but only that it should not be assumed policies will necessarily have a nice clean structure. It might well be that 'nicer' matrices do also exist and maybe also have a good performance.

In this study, our focus is to automatically generate good policies using GA without requiring any human expertise. However, 'human intuition' still can be used before or after the training process and whether or not any such approach would yield better performing policies is not trivial. Hence, initially, we tested a simple idea of setting entries of 'perfect fit' cases to the maximum priority (score) *after training*, so if there is a bin with a remaining size equal to the size of the incoming item, the item is placed into that bin. As an example case study, we applied the idea with UBP(20,5,10,$10^5$). The best policy matrix evolved by GA as reported in Özcan and Parkes (2011) for this generator is converted into a modified policy matrix with 'human intuition' as illustrated in Figure 7. This approach did not alter the performance of the original policy matrix at all and produced the same bin fullness. A similar phenomenon is observed for UBP(6,2,3,$10^5$), as well.

Additionally, we have considered the case of adding human insight *before training*. In the following section, we start the search process using policy matrices which are 'smooth' and 'nice' to see the structures of resultant good policy matrices generated by CHAMP-GA.

Regarding the irregular structure of the automatically-discovered matrices, they are not totally devoid of any visible patterns. For example, consider the good matrix Figure 7(left). Note that a matrix entry of '1' is less than the value of '2' for a new bin and so such entries correspond to decisions that will never be taken (as a new bin is always available). The upper off-diagonal corresponds to entries that lead to a full bin and so is generally preferred. Below this the next 4 off-diagonals correspond to entries that would lead to placing an item into a bin that can never subsequently be used – for example from (size,remaining-space) between (5,6), (5,9), (10,11) and (10,14). In this band the automated

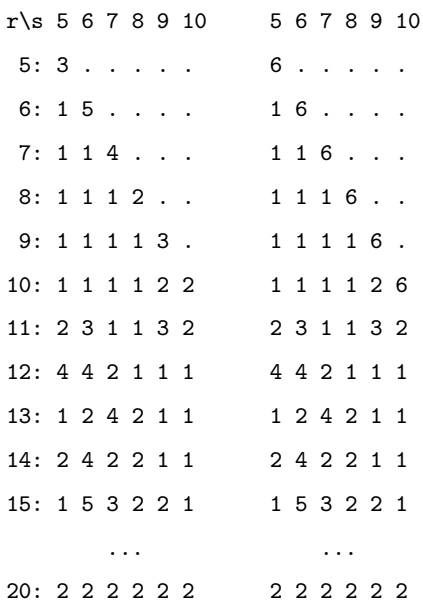



Figure 7: One of the best evolved matrices for UBP(20,5,10,$10^5$) after training on the left and the modified policy matrix by setting the diagonal entries (where remaining size equals to the item size) to the highest score on the right.

search has selected most of the entries to be 1 and so never taken; but note that it did not take all the entries to be 1, leading to decisions to deliberately lose some available space. For example, (size=9,capacity=10) has a value of 2, and then the FF tie-breaking will mean it is preferred to opening a new bin. In manual experiments we found that this is essential – if such cases are not allowed then the performance is severely worse. Hence, there is some regularity in that this 'losing-space band' must be sparsely used, but cannot be removed entirely. Presumably, an important role of the automated search is to discover how best to make such decisions.

### *5.2. Genetic Algorithms for Policy Matrix Generation Using a Different Initial Population*

We have repeated the previous set of experiments with a different population initialisation scheme, specifically by using a FF policy matrix. This modified GA framework will be referred to as $GA_{FFinit}$ which utilises the human insight before training by starting the search from a human designed heuristic. After applying the training and test sessions as described before, the achieved results show that $GA_{FFinit}$ performs worse than the original GA framework ($GA_{Original}$). The results can be seen in Table 5. Also best policy matrices of some instance classes, achieved during the evolution are shown in Figure 8. A Wilcoxon signed-rank test with a confidence level of 99% is conducted using two GA frameworks. Table 3 shows that $GA_{Original}$ performs significantly better than $GA_{FFinit}$ over all the instance generators except the instances generated by UBP$(30, 4, 25, 10^5)$ and UBP$(75, 10, 50, 10^5)$.

Looking at the figures one can immediately conclude that the under-performance of $GA_{FFinit}$ might be due to the fact that FF initialisation scheme generates bias and leads the entire population to some local optima were the population is converged. This explains why the majority of active entries in policy matrices of Figure 8 have the score value 1 (coloured black). Looking at the FF policy matrix in Figure 6, it is easy to notice that all the active entries have their values set to 1. It seems that the cycle of evolution has not been able to emerge

Table 3: Performance comparison between $GA_{Original}$ and $GA_{FFinit}$ over 101 trials based on the Wilcoxon signed-ranked test.

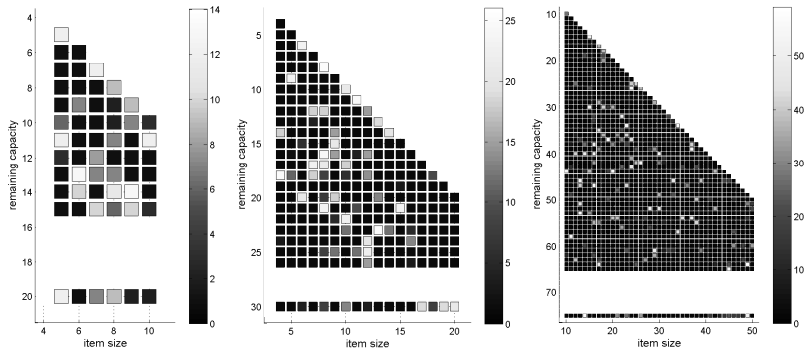
| | Original | vs | FFinit |
|---|---|---|---|
| UBP$(6, 2, 3, 10^5)$ | | $-$ | |
| UBP$(15, 5, 10, 10^5)$ | | $\succeq$ | |
| UBP$(20, 5, 10, 10^5)$ | | $\succ$ | |
| UBP$(30, 4, 20, 10^5)$ | | $\succ$ | |
| UBP$(30, 4, 25, 10^5)$ | | $\prec$ | |
| UBP$(40, 10, 20, 10^5)$ | | $\succ$ | |
| UBP$(60, 15, 25, 10^5)$ | | $\succ$ | |
| UBP$(75, 10, 50, 10^5)$ | | $\prec$ | |
| UBP$(80, 10, 50, 10^5)$ | | $\succ$ | |
| UBP$(150, 20, 100, 10^5)$ | | $\succ$ | |

from this local optima.

High quality policy matrices discovered by $GA_{FFinit}$ for different problem distributions are less 'spiky' than the ones discovered by $GA_{Original}$. However, $GA_{Original}$ performs better than $GA_{FFinit}$. Hence, this set of experiments indicate that there might be a trade-off between the 'smoothness'/'roughness' of the matrix structure and the performance.

### *5.3. Evolving Binary Policy Matrices*

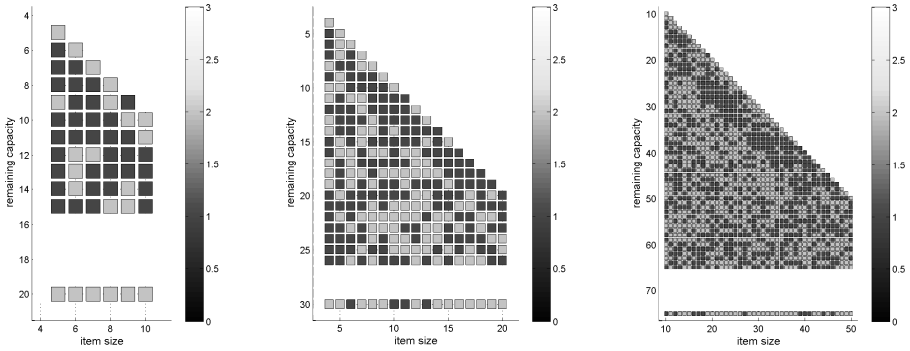
In our next set of experiments, we have devised another variant of the original GA framework ($GA_{Original}$). This variant is particularly considered to investigate the effect of the value on the parameter, $w_{max}$, that is, the upper bound of the scores that an active matrix entry can take. In the previous study (Özcan and Parkes, 2011) $w_{max}$ is set to be the largest number of active entries in each column. We obviously need no more than $n$ different score values to distinguish between $n$ entries. However, we might not need such high a resolution. Also, a high value for $w_{max}$ corresponds to a larger and potentially redundant landscape. Therefore, reducing $w_{max}$ to smaller values may speed up the learn-




(a) UBP$(20, 5, 10, 10^5)$ (b) UBP$(30, 4, 20, 10^5)$ (c) UBP$(75, 10, 50, 10^5)$

Figure 8: The best policy matrices achieved by $GA_{FFinit}$ for selected problem generators

ing process and allow further analysis of the proposed approach. To this effect, $w_{max}$ is set to its minimum value ($w_{max} = 2$) resulting in a binary policy matrix where the active entries take either 1 or 2. That is why we will refer to this variant as $GA_{Binary}$. All other parameter settings for $GA_{Binary}$ is identical to $GA_{Original}$ (Table 1). Similar to previous experiments, the result achieved by this framework is presented in Table 5 and the best policy matrices for some instance generators, achieved during the evolution are shown in Figure 9.




(a) UBP$(20, 5, 10, 10^5)$ (b) UBP$(30, 4, 20, 10^5)$ (c) UBP$(75, 10, 50, 10^5)$

Figure 9: The best policy matrices achieved by $GA_{Binary}$ framework for selected problem generators

It is interesting to observe that by reducing $w_{max}$ (i.e. setting $w_{max} = 2$) slightly better results are achieved compared to the original framework ($GA_{Original}$). This confirms the existence of redundancies in the matrix landscape and its dependency on the value that we choose for $w_{max}$. Thus, a lower resolution for the range of the values that policy matrix active entries can take can actually be reduced without loss of quality. This is confirmed by employing a Wilcoxon signed-ranked test within a confidence interval of 99% and comparing the average performance of the two GA frameworks (Table 4). Similarly, the average performance comparison also confirms that $GA_{Binary}$ performs significantly better than $GA_{FFinit}$ on the instances from all generators except those generated from UBP$(75, 10, 50, 10^5)$.

Table 4: Performance comparison between $GA_{Original}$, $GA_{Binary}$ and $GA_{FFinit}$ over 101 trials based on the Wilcoxon signed-ranked test.

| | Original | vs | Binary | vs | FFinit |
|---|---|---|---|---|---|
| UBP$(6, 2, 3, 10^5)$ | | $-$ | | $-$ | |
| UBP$(15, 5, 10, 10^5)$ | | $\succeq$ | | $-$ | |
| UBP$(20, 5, 10, 10^5)$ | | $\prec$ | | $\succ$ | |
| UBP$(30, 4, 20, 10^5)$ | | $\prec$ | | $\succ$ | |
| UBP$(30, 4, 25, 10^5)$ | | $\prec$ | | $\succ$ | |
| UBP$(40, 10, 20, 10^5)$ | | $\succ$ | | $\succ$ | |
| UBP$(60, 15, 25, 10^5)$ | | $\succ$ | | $\succ$ | |
| UBP$(75, 10, 50, 10^5)$ | | $\prec$ | | $\prec$ | |
| UBP$(80, 10, 50, 10^5)$ | | $\succ$ | | $\succ$ | |
| UBP$(150, 20, 100, 10^5)$ | | $\succ$ | | $\succ$ | |

Although the representation is slightly changed and binary values are used for the parameters of the heuristics generated, binary values are dispersed all around within the best policy matrices found by $GA_{Binary}$. The 'spiky' structure of the policy matrices still remains as it can be seen in Figure 9. However, note that we do not exclude here that some policies may still have some 'nice' structure and deliver good, though maybe reduced, performance at the same

time.

Table 5: Performance comparison of heuristics (BF: best fit, FF: first fit, WF: worst fit, Harmonic (Lee and Lee, 1985)) and various policy matrix generating GA frameworks based on the percentage bin fullness averaged over 101 trials (instances generated by different UBPs). A bold entry indicates the best result and approach for the corresponding instance generator.

| Method | UBP$(6, 2, 3, 10^5)$ | UBP$(15, 5, 10, 10^5)$ | UBP$(20, 5, 10, 10^5)$ | UBP$(30, 4, 20, 10^5)$ | UBP$(30, 4, 25, 10^5)$ | UBP$(40, 10, 20, 10^5)$ | UBP$(60, 15, 25, 10^5)$ | UBP$(75, 10, 50, 10^5)$ | UBP$(80, 10, 50, 10^5)$ | UBP$(150, 20, 100, 10^5)$ |
|---|---|---|---|---|---|---|---|---|---|---|
| BF | 92.30 | 99.62 | 91.55 | 96.84 | 98.38 | 90.23 | 92.55 | 96.08 | 96.39 | 95.82 |
| FF | 92.30 | 99.55 | 91.54 | 96.68 | 97.93 | 90.22 | 92.55 | 95.91 | 96.29 | 95.64 |
| WF | 91.70 | 86.58 | 90.54 | 88.61 | 84.10 | 88.66 | 90.80 | 87.94 | 89.25 | 87.73 |
| Harmonic | - | 74.24 | 90.04 | 73.82 | 74.21 | 89.10 | 85.18 | 71.59 | 72.96 | 71.97 |
| $GA_{Original}$ | **99.99** | **99.63** | 98.18 | 99.41 | 98.39 | **96.99** | **99.68** | 98.22 | **98.54** | **97.88** |
| $GA_{FFinit}$ | **99.99** | 99.61 | 98.15 | 99.10 | 99.25 | 96.05 | 98.28 | 98.43 | 97.87 | 96.92 |
| $GA_{Binary}$ | **99.99** | 99.61 | **98.42** | **99.58** | **99.55** | 96.75 | 96.96 | **98.45** | 98.46 | 97.63 |

*5.3.1. Can roughness be reduced?*

In order to observe whether smoothing out the roughness in the matrices would help, we introduced a *roughness* measure which is multiplied by the fitness value. The roughness measure merely describes how spiky the policy matrix is in terms of the scores associated to active entries. Algorithm 2 is a pseudo code of the roughness measurement method. The main idea is to count the number of occasions in which the score values along each row and column varies. The count is then normalized by the total number of active entries in the matrix, resulting in the *frequency of spikes* within the matrix. Including the roughness measure in fitness calculation yields in an evolutionary process which favours smoother matrices with a less spiky structure.

Since $GA_{Binary}$ is the best performing GA framework so far, it is chosen to test whether the roughness measure affects the performance of the GA framework. Thus, another round of experiments is performed. In this new round of

experiments, $w_{max} = 2$, the population is initialized randomly and the roughness measure is included in the fitness value. We refer to this framework as $GA_{Binary|Roughness}$.

Figure 10 illustrates the influence of the roughness metric in the resultant policy matrices for different termination criteria on UBP$(75, 10, 50, 10^5)$. GA using an objective function embedding the roughness metric still obtains high quality policy matrices which are significantly better than the human designed heuristics. For 200, 400, 600 and 800 iterations, average percentage bin fullness changes from 96.80%, 97.27%, 97.49% and 98.20%, respectively, while the roughness is 30.53, 24.89, 21.23 and 19.01. On the other hand, BF and FF achieves 96.08% and 95.91% with a roughness of 28.97 and 21.62, respectively. Although increasing the maximum number of iterations improves both average bin fullness and roughness with the suggested objective function, still it cannot achieve the same average performance of spiky/rough policy matrix which is 98.45% for UBP$(75, 10, 50, 10^5)$. Hence, some kind of irregularity or roughness appears to be essential which confirms the observations in Section 5.2; though further research is needed to see if other measures of roughness may lead to less irregular matrices.

*5.3.2. Fitness landscape analysis*

Fitness landscape analysis has been performed on two separate problem instance generators of UBP$(6, 2, 3, 10^5)$ and UBP$(15, 5, 10, 10^5)$. The $GA_{Binary}$ variant is used in this part of our studies, that is, the active entries of the policy matrix take values of 1 or 2. The experiment consists of evaluating the fitness value of all possible binary individuals (policies) for UBP$(6, 2, 3, 10^5)$ and a randomly sampled subset of all possible individuals for UBP$(15, 5, 10, 10^5)$.

For UBP$(6, 2, 3, 10^5)$, the entire fitness landscape is sampled as the landscape includes reasonably low number of possible individuals. However, in case of UBP$(15, 5, 10, 10^5)$ where the total number of individuals in the landscape is around $2^{27}$, and so full coverage of the landscape is intractable. This is due to the fact that some individuals represent low quality matrices and it takes much

**Algorithm 2:** The roughness computation

```
In : W;
Out : roughness;
//column-wise roughness measurement;
for each active entry in column s do
    for each active entry in row r do
        if subsequent scores in a column are not equal then
            n++;
        end
    end
end
//row-wise roughness measurement;
for each active entry in row r do
    for each active entry in column s do
        if subsequent scores in a column are not equal then
            n++;
        end
    end
end
roughness = n / (2×(# active entries));
```

longer time to evaluate them; poorer solutions tend to open many more bins, and processing these increases the run-time. Consequently, full coverage of the entire landscape is avoided by sampling the landscape according to a uniform random distribution with a probability equal to $10^{-4}$. Moreover, redundant individuals are eliminated. A redundant individual consists of a column in which the active entries are all equal to 2. The reason it is redundant is the fact that an individual with exact same entries in all other columns and the value of 1 for the column under inspection will result with the same fitness value. The total number of possible individuals after redundancy elimination

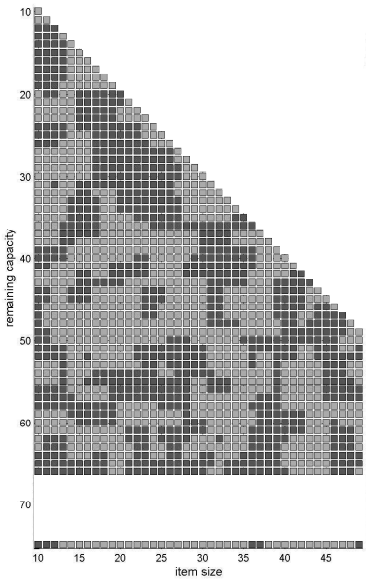



Figure 10: The best policy matrix obtained by $GA_{Binary|Roughness}$ on UBP$(75, 10, 50, 10^5)$.

and prior to random sampling for UBP$(15, 5, 10, 10^5)$ is $78,129,765$. After all the pre-processing, the individuals (policies) are evaluated by the bin packing program.

The Fitness-Distance Correlation (FDC) as well as Correlation Length (CL) is then calculated for each UBP. The FDC measure, proposed in Jones and Forrest (1995), is a measure of search difficulty. Suppose that $F = \{f_1, f_2, \ldots, f_n\}$ is a set of $n$ individual fitnesses. Furthermore, suppose that $D = \{d_1, d_2, \ldots, d_n\}$ is the set of the distances of each solution to its closest global maxima. The fitness distance correlation coefficient is then computed by the following equation.

$$r = \frac{C_{FD}}{\sigma_F \sigma_D} \quad where \quad C_{FD} = \frac{1}{n} \sum_{i=1}^{n} (f_i - \bar{f})(d_i - \bar{d}) \tag{1}$$

In Eq.1, $\bar{f}, \bar{d}, \sigma_F$ and $\sigma_D$ are the mean of all fitness values, the average of distance values and the standard deviations of fitness and distance values respectively. The FDC value has a range of $[-1, 1]$. Considering a maximisation problem, a value closer to 1, indicates that the underlying problem is a mislead-

Table 6: FDC and CL measures

| ID | UBP$(6, 2, 3, 10^5)$ | UBP$(15, 5, 10, 10^5)$ |
|---|---|---|
| FDC | 0.176 | -0.516 |
| CL | 0.565 | 9.369 |

ing one; the search tends to be led away from the global optimum. FDC values close to $-1$ indicate easy and straightforward problems while FDC values close to 0 the prediction is indeterminate.

The Correlation Length (CL) measures the ruggedness of the fitness landscape and is based on autocorrelation. The autocorrelation value, calculated using Eq.2, is a measure of the correlation between two points separated by $i$ random steps.

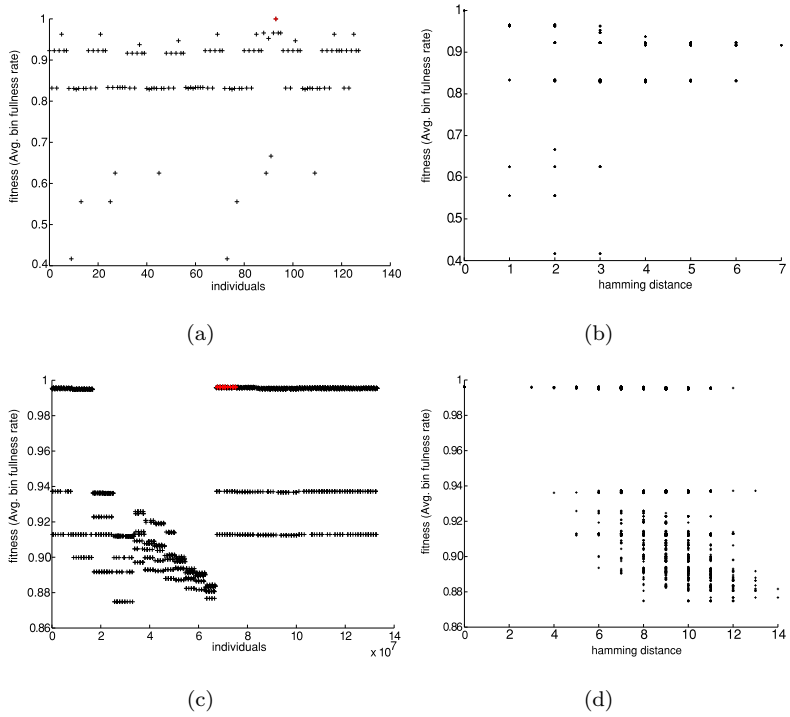
$$\rho_s = \frac{\sum_{i=1}^{n-s}(f_i - \bar{f})(f_{i+s} - \bar{f})}{\sum_{i=1}^{n}(f_i - \bar{f})^2} \tag{2}$$

In Eq.2, $s$ is the step size. The CL value of a landscape ($\ell$) gives the largest distance, in terms of the number of steps, for which there still is a correlation between the starting and the ending point (Eq.3). A high value for $\ell$ implies a smooth landscape whereas a low correlation length means a rugged landscape.

$$\ell = -\frac{1}{ln(|\rho_1|)} \tag{3}$$

The results in Table 6 indicate that UBP$(6, 2, 3, 10^5)$ is a difficult problem for the evolutionary algorithm with an FDC of 0.176, while UBP$(15, 5, 10, 10^5)$ is an easy problem for the evolutionary algorithm with an FDC of -0.516. This is potentially because the fitness landscape is rugged for UBP$(6, 2, 3, 10^5)$, while it is not for UBP$(15, 5, 10, 10^5)$ considering the CL values.

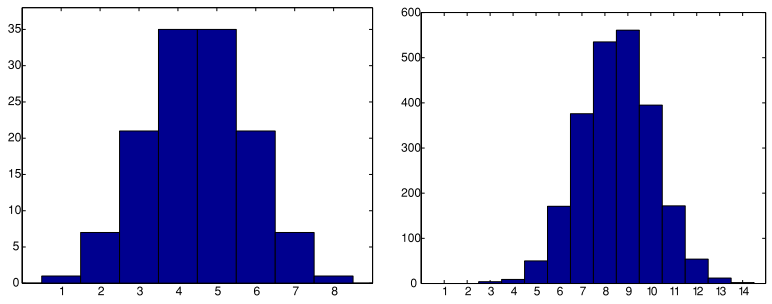
Figure 11(a) and 11(c) provides the fitness value of each sampled individual in the search landscape for UBP$(6, 2, 3, 10^5)$ and UBP$(15, 5, 10, 10^5)$, respectively. The search for the optimum policy for UBP$(6, 2, 3, 10^5)$ is like a search for a needle in the hay stack, since there is only a single optimum pol-




(a) (b)

(c) (d)

Figure 11: Fitness landscape analysis results for (a), (b) UBP$(6, 2, 3, 10^5)$ and (c),(d) UBP$(15, 5, 10, 10^5)$. Figure 11 (a), (c) provide the plot of each sampled binary encoded policy (individual) and its fitness, while (b), (d) are the scatter plot of the fitnesses of policies (individuals) in terms of average bin fullness and their hamming distances to/from the closest best solution in the sample of matrices for the relevant UBP.




(a) UBP$(6, 2, 3, 10^5)$ (b) UBP$(15, 5, 10, 10^5)$

Figure 12: The histogram of the hamming distances between individuals and the closest optimum. The $x$-axis represents the hamming distance and the $y$-axis refers to the number of individuals.

icy in the search space as shown in Figure 11(a), while the search landscape of UBP$(15, 5, 10, 10^5)$ contains multiple best solutions as illustrated in Figure 11(c). Figure 11(b) and 11(d) demonstrate the correlation between fitness value and the hamming distance between individuals of UBP$(6, 2, 3, 10^5)$ and UBP$(15, 5, 10, 10^5)$ and the closest best solution, respectively. Moreover, Figure 12(a) and 12(b) provides a histogram illustrating the number of individuals with a specific hamming distance to the closest best solution for UBP$(6, 2, 3, 10^5)$ and UBP$(15, 5, 10, 10^5)$, respectively. The plots reemphasize the fact that the fitness landscape is rugged for UBP$(6, 2, 3, 10^5)$ and there are disconnected plateaus, while the search landscape is smoother for UBP$(15, 5, 10, 10^5)$. Moreover, there is not much correlation between the distance to the optimal/best solution and fitness for UBP$(6, 2, 3, 10^5)$, while there is a correlation for UBP$(15, 5, 10, 10^5)$.

## 6. Conclusion

In this study, we have presented a framework, and clarified the associated methodology, which can be used for creating heuristics via many parameters (CHAMP). Under CHAMP, a heuristic corresponding to a policy chooses the highest value option while making decisions. This framework is used to generate and search for heuristics to solve some online packing problems with a

genetic algorithm (CHAMP-GA). The resulting policies are specialised to the distributions and are much more effective, and have a quite different structure, than those the existing general-purpose ones. An important lesson might be that, in complex situations, our intuition about the nature of good heuristics can be quite sub-optimal, and that search-based generation can give significantly better results, as well as (potentially) requiring less intervention by an expert; hopefully, this may ultimately reduce total cost-of-ownership of such systems, allowing wider usage. This is not to say that there are no special UBP instance generators in which a human intuition might provide better solutions, but that CHAMP can provide a tool that allows policy discovery more generally, and is capable of getting good or very good solutions, with relatively low computational effort.

CHAMP-GA discovered 'spiky' matrices (policies) for some UBP instance generators. The genetic programming approach as provided in (Burke et al., 2006, 2007b) and human designed policies for online bin packing implicitly make the assumption that 'cleanly structured nice' policies are the best solutions. However, evidence presented by this work shows that this does need not to be true in general. So, no assumptions on the structure of solutions should be made, even implicitly, while creating policies whether by genetic programming, another heuristic optimisation method or hand crafting.

The fitness landscape analysis of CHAMP-GA on some instance generators has confirmed that the search space of policies are sometimes rugged and sometimes they are not with neutral regions depending on the generator dealt with. Regardless, it has been observed that if we use a policy defined by a simple matrix of score values, then a standard GA approach can produce policies tuned to the distribution of the instances under consideration and substantially outperforming the generic heuristics such as first and best fit.

Obvious avenues for future research are: To study variants of the online bin packing (for example, the distribution of item sizes is not uniform); improve the search methods used to discover good matrices; and apply the general approach to other problem domains.

**Acknowledgements**

This work was supported in part by the EPSRC, grant EP/F033214/1.

**References**

Asta, S., Özcan, E., 2015. A tensor analysis improved genetic algorithm for online bin packing, in: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA. pp. 799–806.

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

Asta, S., Özcan, E., Parkes, A.J., Etaner-Uyar, S., 2013b. Generalizing hyper-heuristics via apprenticeship learning, in: Middendorf, M., Blum, C. (Eds.), Evolutionary Computation in Combinatorial Optimization. Springer Berlin Heidelberg. volume 7832 of *Lecture Notes in Computer Science*, pp. 169–178.

Bernstein, Y., Li, X., Ciesielski, V., Song, A., 2004. Multiobjective parsimony enforcement for superior generalisation performance, in: Greenwood, G. (Ed.), Proceedings of the IEEE Congress on Evolutionary Computation (CEC2004), pp. 83–89.

Bottou, L., 2012. Stochastic gradient descent tricks, in: Montavon, G., Orr, G.B., Müller, K.R. (Eds.), Neural Networks: Tricks of the Trade: Second Edition. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 421–436.

Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R., 2013. Hyper-heuristics. J Oper Res Soc 64, 1695–1724.

Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S., 2003. Hyper-heuristics: An emerging direction in modern search technology, in: Glover, F., Kochenberger, G. (Eds.), Handbook of Metaheuristics. Kluwer, pp. 457–474.

Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R., 2010. A classification of hyper-heuristic approaches, in: Gendreau, M., Potvin, J.Y. (Eds.), Handbook of Metaheuristics. Springer US. volume 146 of *International Series in Operations Research and Management Science*, pp. 449–468.

Burke, E.K., Hyde, M.R., Kendall, G., 2006. Evolving bin packing heuristics with genetic programming, in: Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006), Reykjavik, Iceland. pp. 860–869.

Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R., 2009. Exploring hyper-heuristic methodologies with genetic programming, in: Kacprzyk, J., Jain, L.C., Mumford, C.L., Jain, L.C. (Eds.), Computational Intelligence. Springer Berlin Heidelberg. volume 1 of *Intelligent Systems Reference Library*, pp. 177–201.

Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J., 2007a. 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, ACM, New York, NY, USA. pp. 1559–1565.

Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.R., 2007b. The scalability of evolved on line bin packing heuristics, in: Srinivasan, D., Wang, L. (Eds.), 2007 IEEE Congress on Evolutionary Computation, IEEE Computational Intelligence Society. IEEE Press, Singapore. pp. 2530–2537.

Chakhlevitch, K., Cowling, P., 2008. Hyperheuristics: Recent developments, in: Cotta, C., Sevaux, M., Sörensen, K. (Eds.), Adaptive and Multilevel Metaheuristics. Springer Berlin / Heidelberg. volume 136 of *Studies in Computational Intelligence*, pp. 3–29.

Coffman, Jr., E.G., Garey, M.R., Johnson, D.S., 1997. Approximation algorithms for bin packing: a survey. PWS Publishing Co., Boston, MA, USA. pp. 46–93.

Coffman Jr., E.G., Csirik, J., Galambos, G., Martello, S., Vigo, D., 2013. 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, pp. 455–531.

Coffman Jr, E.G., Galambos, G., Martello, S., Vigo, D., 1999. Bin packing approximation algorithms: Combinatorial analysis, in: Du, D.Z., Pardalos, P. (Eds.), Handbook of Combinatorial Optimization. Kluwer Academic Publishers. volume 1 of *Intelligent Systems Reference Library*, pp. 151–207.

Csirik, J., Woeginger, G., 1998. On-line packing and covering problems, in: Fiat, A., Woeginger, G. (Eds.), Online Algorithms. Springer Berlin / Heidelberg. volume 1442 of *Lecture Notes in Computer Science*, pp. 147–177.

Falkenauer, E., 1996. A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics 2, 5–30.

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

Gittins, J., Glazebrook, K., Weber, R., 2011. Multi-armed Bandit Allocation Indices. Wiley. 2 edition.

Gittins, J.C., 1979. Bandit processes and dynamic allocation indices. Journal of the Royal Statistical Society. Series B (Methodological) 41, pp. 148–177.

Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham, R.L., 1974. Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on Computing 3, 299–325.

Jones, T., Forrest, S., 1995. Fitness distance correlation as a measure of problem difficulty for genetic algorithms, in: Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann. pp. 184–192.

Lee, C.C., Lee, D.T., 1985. A simple on-line bin-packing algorithm. Journal of the ACM 32, 562–572.

Marín-Blázquez, J.G., Schulenburg, S., 2006. Multi-step environment learning classifier systems applied to hyper-heuristics, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA. pp. 1521–1528.

Martello, S., Toth, P., 1990. Lower bounds and reduction procedures for the bin packing problem. Discrete Appl. Math. 28, 59–70.

Özcan, E., Bilgin, B., Korkmaz, E.E., 2008. A comprehensive survey of hyper-heuristics. Intelligent Data Analysis 12, 3–23.

Özcan, E., Parkes, A.J., 2011. Policy matrix evolution for generation of heuristics, in: Proceedings of the 13th annual conference on Genetic and evolutionary computation, ACM, New York, NY, USA. pp. 2011–2018.

Parkes, A.J., Özcan, E., Hyde, M.R., 2012. 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, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 158–169.

Pillay, N., 2012. A study of evolutionary algorithm selection hyper-heuristics for the one-dimensional bin-packing problem. South African Computer Journal 48, 31–40.

Poli, R., Langdon, W.B., McPhee, N.F., 2008. A field guide to genetic programming. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`. (With contributions by J. R. Koza).

Rhee, W.T., Talagrand, M., 1993. On line bin packing with items of random size. Mathematics of Operations Research 18, pp. 438–445.

Richey, M.B., 1991. Improved bounds for harmonic-based bin packing algorithms. Discrete Applied Mathematics 34, 203 – 227.

Ross, P., 2005. Hyper-heuristics, in: Burke, E.K., Kendall, G. (Eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. Springer. chapter 17, pp. 529–556.

Ross, P., Schulenburg, S., Marín-Blázquez, J.G., Hart, E., 2002. Hyper-heurstics: learning to combine simple heuristics in bin-packing problems, in: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, pages 942-948, New York NY, 2002. Morgan Kauffmann Publishers.

Scholl, A., Klein, R., Jürgens, C., 1997. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers & Operations Research 24, 627 – 645.

Seiden, S.S., 2002. On the online bin packing problem. Journal of the ACM 49, 640–671.

Sim, K., Hart, E., 2013. Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model, in: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA. pp. 1549–1556.

Smit, S.K., Eiben, A.E., 2009. Comparing parameter tuning methods for evolutionary algorithms, in: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, USA. pp. 399–406.

Sörensen, K., Glover, F.W., 2013. Metaheuristics, in: Encyclopedia of operations research and management science. Springer US, pp. 960–970.

Tavares, J., Pereira, F., Costa, E., 2008. Multidimensional knapsack problem: A fitness landscape analysis. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 38, 604–616.

Ülker, O., Korkmaz, E., Özcan, E., 2008. A grouping genetic algorithm using linear linkage encoding for bin packing, in: Rudolph, G., Jansen, T., Lucas, S.,

Poloni, C., Beume, N. (Eds.), Parallel Problem Solving from Nature - PPSN X. Springer Berlin / Heidelberg. volume 5199 of *Lecture Notes in Computer Science*, pp. 1140–1149.

Wright, S., 1932. The roles of mutation, inbreeding, crossbreeding and selection in evolution. Proc.Int.Cong.Gen. 1, 356–366.

Yarimcam, A., Asta, S., Özcan, E., Parkes, A., 2014. Heuristic generation via parameter tuning for online bin packing, in: Evolving and Autonomous Learning Systems (EALS), 2014 IEEE Symposium on, pp. 102–108.