CHaMP: Creating Heuristics via Many Parameters

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 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 all item sequence generators used during the experiments.

Keywords: Genetic algorithms; heuristics; packing; decision support systems; learning systems.

Email addresses: sba@cs.nott.ac.uk (Shahriar Asta), Ender.Ozcan@nottingham.ac.uk (Ender Özcan), ajp@cs.nott.ac.uk (Andrew J. Parkes)

1. Introduction

In many situations, decisions must be made despite lack of knowledge of the future allowing the computation of the full effects of the decisions. In such cases, it is usual to have some kind of heuristic 'dispatch policy' to make decisions. Usually, such heuristics are produced by an expert in a domain carefully designing some decision procedure. Often, even an expert requires a great deal of trial and error - though the errors are rarely reported, and so a misleading impression is given suggesting that creation of heuristics is not a time-consuming process. Of course, such difficulties are well-known, and so there have been various attempts to automate the production of heuristics (e.g. for some recent work see Ross (2005); Chakhlevitch and Cowling (2008); Burke et al. (2009)).

In previous work of Özcan and Parkes (2011), an approach for the automatic creation of heuristics is given that might be viewed as a form of parameter tuning (Smit and Eiben, 2009), but applied with a much larger number of parameters than usually considered. The large number of parameters arise from a 'brute force' representation of the heuristic as a matrix covering the various potential decisions, that is, it defines a policy in terms of the 'features' available at each decision point. This is done in the style of an 'index policy' (e.g., Gittins (1979)) in that each potential outcome is given a score separately of other outcomes and the largest score is selected.

In this paper, we particularly study the well-known online bin-packing problem (Coffman et al., 1997; Csirik and Woeginger, 1998), creating a policy that is based on using a (large) matrix[1] of 'heuristic scores'. The policy matrix can be viewed as a heuristic with many parameters. 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.

[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

The primary result (reported in Özcan and Parkes (2011) and Asta et al. (2013)) 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). In this paper, we firstly clarify some potentially confusing and counter-intuitive issues in the appropriate search methodology. We then investigate various parameter settings for the proposed policy matrices, including the upper bound of the scores, various initialization schemes and fitness evaluation measures.

A *fitness landscape analysis* (Wright, 1932; Tavares et al., 2008) was also conducted to better understand the performance of the proposed algorithm in relation with the policy matrix representation. 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 this value often grows exponentially with respect to the size of a problem. Hence, except for small instances, 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 sequence; 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 metaheuristic 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, situtations 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. A related issue is that we will be creating heuristics for specific (stochastic)

generators of item sequences. This might at first seem like it is 'over-tuning', however, our aim is precisely to develop methods that can adapt to specific cases and exploit their properties. 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 gives basic definitions of the bin-packing problem, the instances that we use, and the existing standard heuristics. Section 4 specifies the policy matrix-based CHAMP framework we use in order to define the packing heuristics. Section 4.3 describes the GA search method we use to find good policies. 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. Section 6 gives the landscape analysis. Section 7 summarises our results and their implications, and then discusses future plans.

2. Related Work

We now give a brief review and pointers into the literature of existing work, firstly on bin-packing and then on computer-based methods to help design heuristics.

2.1. Bin-packing

One dimensional (offline) bin packing is a combinatorial optimisation grouping problem, proven to be NP-hard (Garey and Johnson, 1990). This problem involves packing a number of pieces with given sizes into a minimal number of bins, where every bin has the same fixed capacity. In other words, a set of integers must be partitioned into subsets (groups) so that the sum of the integers within a subset does not exceed the capacity. In this generic offline problem, the solution approaches have complete information about the number of pieces and their sizes. Heuristics are commonly used for bin packing Coffman Jr et al. (1999), whenever exact algorithms fail to produce a result in reasonable time for a given problem. There are also other studies addressing the representation issues for bin packing in metaheuristics. For example, Falkenauer (1996) used group encoding for candidate solution representation in the framework of genetic algorithm and combined the grouping

genetic algorithm with local search based on the branch and bound reduction algorithm in Martello and Toth (1990). Ülker et al. (2008) presented a linear linkage encoding to overcome the redundancy in the group encoding representation within another grouping genetic algorithm framework. More on bin packing can be found in Coffman et al. (1997); Csirik and Woeginger (1998).

Online bin packing problem is a well known variant in which the pieces arrive sequentially and at each step, a packing decision has to be made before the next item size is revealed (Seiden, 2002; Lee and Lee, 1985; Richey, 1991). The decision is made under incomplete information about the number of pieces and their sizes, and results in putting the current item into an already open bin or opening a new bin.

2.2. Existing methods to discover heuristics

Hyper-heuristics provide search and optimisation frameworks which allow the exploration of heuristics space for solving complex problems (Burke et al., 2003; Ross, 2005; Özcan et al., 2008; Chakhlevitch and Cowling, 2008; Burke et al., 2013). There are two main classes of hyper-heuristics; methodologies that *select* or *generate* low level heuristics (Burke et al., 2010). Genetic Programming (GP) is an Evolutionary Algorithm that searches the space of computer programs. It has been applied to many different challenging problems (Poli et al., 2008). GP has been frequently used as a generation hyper-heuristic for the automated design of heuristics (Burke et al., 2009).

In Ross et al. (2002), a learning classifier system of the Michigan type XCS was trained to generate a hyper-heuristic for solving unseen bin packing problem instances. The system learns how to choose a low level heuristic to iteratively construct a solution starting from an initial state towards a final state, in which no item is left for packing. The authors reported that XCS performed well across a large collection of data. In this study, the number of training instances used was much larger than the test cases.

Particularly relevant is the work in Burke et al. (2006) that used GP to evolve heuristics to decide the choice of bin for packing a given item. The results showed that the first fit heuristic could be generated by the genetic programming approach. The authors reported that the code-bloat (Bernstein et al., 2004) existed in their genetic programming. Moreover, they confirmed the redundancy in the representation, as GP generated four trees with depth of 2 showing similar performances to the first fit heuristic. Subsequent work in Burke et al. (2007b) obtained new heuristics after a training phase using

genetic programming and tested these online bin packing heuristics on randomly generated problem instances. The authors investigated the behavior of the computer generated heuristics as the problem size increases against different sizes of training problem instances. As expected, they have observed that the performance of generated heuristics improves as the size of training problems and test problems increase. The new heuristics produced a competitive performance to the best fit heuristic. The same authors (Burke et al., 2007a) further studied the trade off between performance and level of generality of a heuristic generated by genetic programming for online bin packing. The authors have illustrated that indeed there is a trade-off and the representative problem instances used during the training phase is crucial.

3. Online Bin Packing Problem

In online one dimensional bin packing, each bin has a capacity $C > 1$ and each item size is a scalar in the range $[1, C-1]$. More specifically, each item can be chosen from the range $[s_{min}, s_{max}]$ where $s_{min} > 0$ and $s_{max} < C$. 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.

The uniform 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 number of total items. For example, UBP$(15, 5, 10, 10^5)$ is a random instance generator and represents a class of problem instances. Each 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.

It is crucial to note the two primary different ways in which random instance generators are used in the OR. A common usage is to create a gen-

erator and then then create around a few dozen instances which then become individual public benchmarks, and methods are tested by giving results on those individual. In our case, the aim is to create heuristics that work on average across all instances from the generator. (Hence, for example, we believe it would not serve any useful purpose to place our specific training instance sequences on a website.) A related note is that it is important to distiguish two quite different meanings of 'instance': either a specific generator, or a specific sequence of items. An instance in the sense of a generator generally contains a Psuedo-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 items.

There are well established heuristics for this problem among which 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) provide a worst-case performance ratio better than the other heuristics. Assuming that the size of an item is a value in (0,1], 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.

Although we often refer to the choices for UBP as instances it should be remembered that they are instances of 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 generated. Indeed, this is the case when we test our approach as it will be seen in the coming sections.

There are various criteria with which the performance of a bin packing solution can be evaluated. Some of these are enlisted below.

Bins-Used, B: The number of bins that are used. B is an integer value which tends to increase as larger number of items (N) are considered.

Average-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.$F_{af} = 1/B \sum_t f_t$

Average-Generic-Fullness, F_{gf}: This value gives some insight into the variation of resulting fullness between bins. $F_{gf} = 1/B \sum_t f_t^2$

Average-Perfection, F_{ap}: This measure is an indication of how successful the heuristic is in packing the bins perfectly. $F_{ap} = 1/B \sum_{t,f_t=1} f_t$

In our study, average bin fullness (F_{af}) is considered as the fitness (evaluation/objective) function.

4. Solution Methodology

4.1. Policy Matrix Representation

A packing policy can be defined by a matrix of heuristic values (called policy matrix). That is, we have a matrix structure in which for each pair (r,s) a score $W_{r,s}$ is provided which 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 (Algorithm.1). A feasible open bin is an open bin with enough space for the current item size, say $r \geq s$ and includes the always-available new empty bin. The integer scores are chosen from a specific range $W_{r,s} \in [w_{min}, w_{max}]$.

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$(15, 5, 10, 10^5)$ as demonstrated in Figure 1. In this matrix, $w_{min} = 1$ and $w_{max} = 7$, which is the maximum number of active items in a column. During the packing process, an item with a size of

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
```

5 arrives. The column 5 in the figure corresponds to this item. 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 5 and 1 respectively. The score associated to the empty bin is 2. The highest score is thus 5 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 15 (empty bin) was 7 instead of the current value of 2, 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). 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 column 8. Also assume that in addition to the new 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 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.

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

Figure 1: An example of a policy matrix for UBP$(15, 5, 10, 10^5)$

4.2. 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.

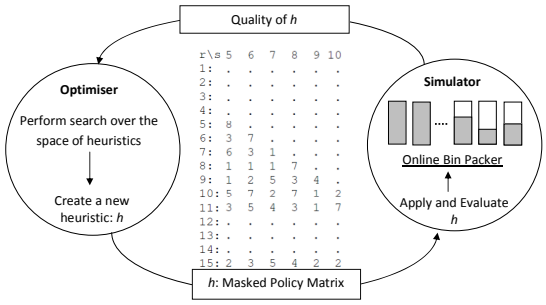


Figure 2: CHAMP framework for the online bin packing problem.

4.3. 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 evaluation. Each individual is evaluated by applying it to the bin packing problem instance as was shown in Algorithm 1 and the fitness value is one (or more) of the measures in Section 3. The settings for the GA optimiser is given in Table.1.

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	Traditional
Mutation Rate	$1/ChromosomeLength$
No. of trials	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 total number of bins used while solving each training case is accumulated and then saved as the fitness of the individual into 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 has been considered for experiments. Moreover, several variants of the policy matrix evolution scheme has been considered each differing with others in the initialization scheme, and the upper bound for score range (w_{max}). In Section 5 the results of applying the variants of the framework to problems is reported.

5. Computational Experiments

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 also seeding the initial population with the first fit policy. In the second and third set of experiments, we looked into the performance of binary valued policy matrices. Finally, (see the next section), we performed landscape analysis on the online bin packing problems.

5.1. Experimental Methodology and Design

Generally, our experiments consists of training and test phases. In the training phase, the GA framework is trained on each of the problem instance generators where a training session consists of a single run. When training is finished the best policy matrix for each UBP is stored and is later used for testing. 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 evolution process, 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 fitness function. A specific seeding scheme has been employed which renders each training session different from other sessions. That is, there exists a fixed length list of seed values stored in a file. When the training starts, the first seed value is read from the file and is fed to the bin packing program as the seed value. This is repeated for a fixed number of calls (m) to the bin packing program ($m = 10$ in our experiments). Subsequently, the next number in the list is chosen as the seed value for the upcoming m calls to the bin packing program. 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. In the test phase, for a given problem instance generator the bin packer is executed for 101 trials (also different seed values), utilizing the best policy matrix achieved during training which corresponds to the UBP instance generator. The 'peculiarity' of always using the same seed for the training instances is the same as the 'Common Random Numbers' used as a techniques to reduce variance in simulation and sampling studies. We did explore change the seed within each generation - so that the training instance(s) would change, 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).

The reason for ever performing a single run 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, and so train on the largest sequence 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 run should be one run with 10^5 items, or R runs 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;

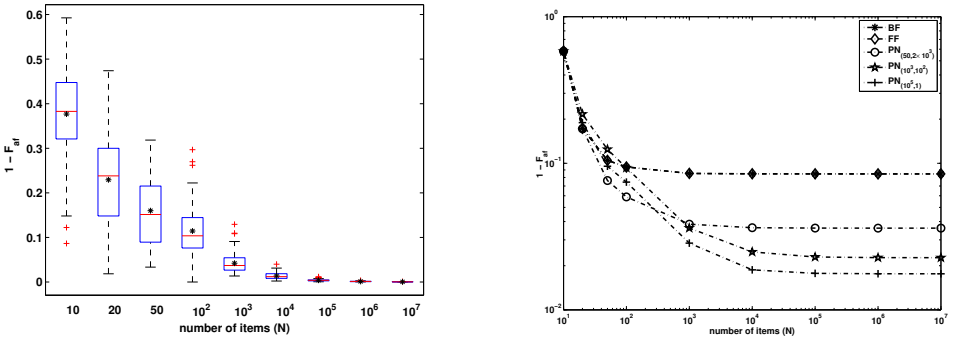
r\s	1	2	3	4	5	6	7	8	9
1:	2	.	.	.	.	.	.	.	.
2:	1	2	.	.	.	.	.	.	.
3:	1	1	2	.	.	.	.	.	.
4:	1	1	1	2	.	.	.	.	.
5:	1	1	1	1	2	.	.	.	.
6:	1	1	1	1	1	2	.	.	.
7:	1	1	1	1	1	1	2	.	.
8:	1	1	1	1	1	1	1	2	.
9:	2	2	2	2	2	2	2	2	2

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

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 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 for 2×10^3, 10^2 and 1 time(s) generating 50, 10^3 and 10^5 items, respectively and tested the best policy generated by GA, FF and BF on 101 instances of UBP$(9, 1, 9, N)$



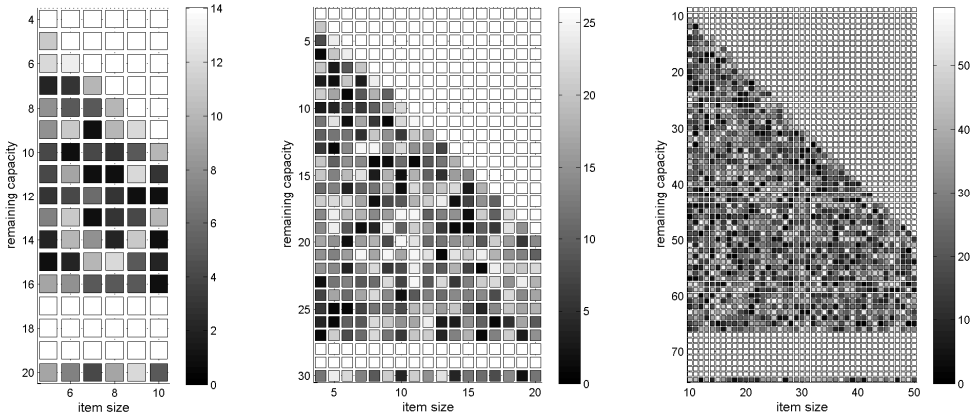
(a) (b)

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 2×10^3, 10^2 and 1 time(s), respectively, on UBP$(9, 1, 9, N)$ for various N. The x-axis uses a log-scale.

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 training with a single long run is viable.

5.2. Analysis of Genetic Algorithms for Policy Matrix Generation

Before starting our experiments, an extensive analysis of the GA methodology proposed in Özcan and Parkes (2011) is performed by applying it to a wider range of bin packing instance generators (Table 2). The original GA framework is referred to as $GA_{Original}$ throughout this paper. During training, $GA_{Original}$ is used to find an optimal policy matrix for each instance generator (UBP). An example of the best policy matrices of some selected



(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 matrix achieved by the framework proposed in Özcan and Parkes (2011)

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$(150, 20, 100, 10^5)$

instance generators achieved in the training phase is demonstrated in Figure 5. The average, minimum and maximum objective values achieved for each instance generator during the test phase can be seen in Table 5.

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. This is understandable as the Harmonic algorithm is designed for the worst case over all sequences, rather

than the average case.

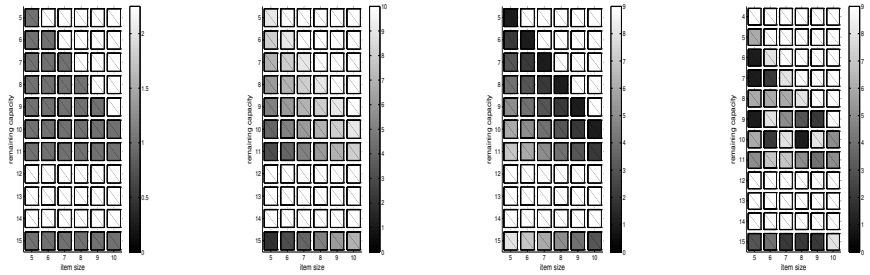
FF, BF and WF heuristics can also be represented as policy matrices. Figure 6 shows a policy matrix of each heuristic for the instance generator UBP$(20, 5, 10, 10^5)$. It is easy to note that the policy matrices corresponding to FF, BF and WF heuristics are smooth matrices. The score values change monotonically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

However, we have only demonstrated that some of the best matrices can be 'spiky'. 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. Hence, in the following section, we start the search process using policy matrices which are 'smooth' and 'nice' to see the structures of good policy matrices generated by GA-CHAMP.

5.3. Genetic Algorithms for Policy Matrix Generation Using a Different Initial Population

We have repeated the previous set of experiments with a different population initialization scheme, specifically by using a FF policy matrix. This



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

Figure 6: Smooth FF, BF and WF policy matrices corresponding to an instance of UBP$(15, 5, 10, 10^5)$ compared to the spiky structure of the (much better performing) best policy matrix achieved by $GA_{Original}$ for an instance of the same UBP.

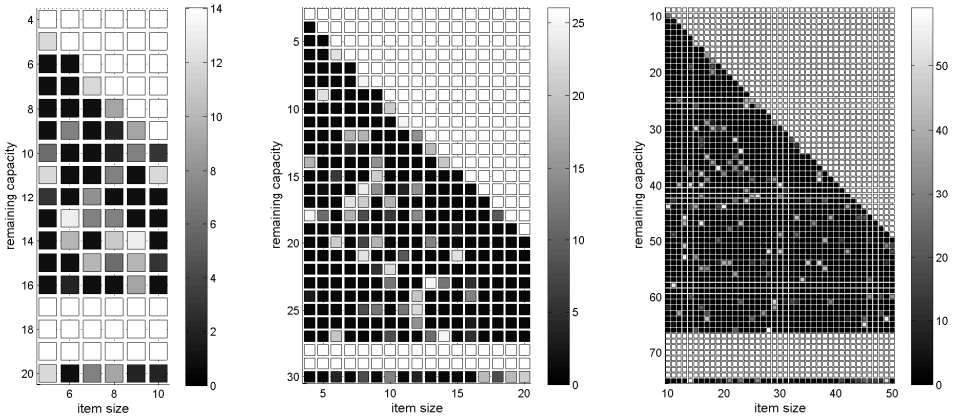
modified GA framework will be referred to as GA_{FFinit}. After applying the training and test sessions as described before, the achieved results show that GA_{FFinit} performs worse than the original GA framework ($GA_{Original}$). The results can be seen in Table 5. Also best policy matrices of some instance classes, achieved during the evolution are shown in Figure 7. A paired t-test, with a confidence level of 0.01, is conducted on the performances of the two GA frameworks (Table 3). The paired test suggests that $GA_{Original}$ outperforms GA_{FFinit} in all the instance generators except instances generated by UBP$(30, 4, 25, 10^5)$ and UBP$(75, 10, 50, 10^5)$. In Table 3, the notation $>$ / $<$ ($\geq$ / $\leq$) denotes a (slightly) better/worse performance of the left hand side algorithm and that this difference in performance is (not) statistically significant within 99% confidence interval based on the paired t-test.

Looking at the figures one can immediately conclude that the underperformance of GA_{FFinit} might be due to the fact that FF initialization 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 7 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 from this local optima.

High quality policy matrices discovered by GA_{FFinit} for the problem instances are less 'spiky' than the ones discovered by $GA_{Original}$. However, $GA_{Original}$ performs better than GA_{FFinit}. Hence, this set of experiments in-

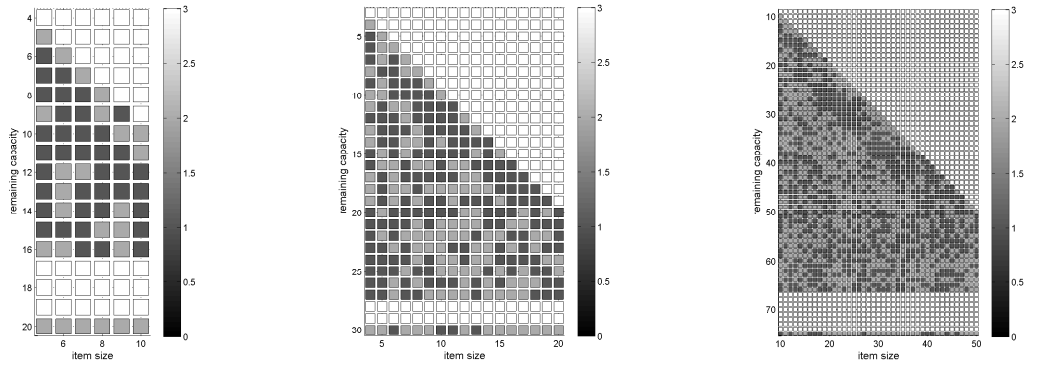
Table 3: Statistical comparison between $GA_{Original}$ and GA_{FFinit} over 101 runs.

	Original vs FFinit
UBP$(6, 2, 3, 10^5)$	$-$
UBP$(15, 5, 10, 10^5)$	$\geq$
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$(150, 20, 100, 10^5)$	$>$



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

Figure 7: The best policy matrices achieved by GA_{FFinit}



(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_{Binary} framework

dicate that there might be a trade-off between the 'smoothness' of the matrix structure and the performance.

5.4. Evolving Binary Policy Matrices

In our next step 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 learning 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 represented in Table 5 whereas best policy matrices for instances of some instance generators, achieved during the evolution are shown in Figure 8.

It is interesting to observe that by reducing w_{max} (i.e. setting $w_{max} =$

Table 4: Statistical comparison between $GA_{Original}$, GA_{Binary} and GA_{FFinit} over 101 runs.

	Original	vs	Binary	vs	FFinit
UBP$(6, 2, 3, 10^5)$		$-$		$-$	
UBP$(15, 5, 10, 10^5)$		$\geq$		$-$	
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$(150, 20, 100, 10^5)$		$>$		$>$	

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 paired test with a 0.01 confidence level to compare the performance of the two GA frameworks (Table 4). A similar statistical comparison also confirms that GA_{Binary} outperforms GA_{FFinit} on instances of all generators except those generated by UBP$(75, 10, 50, 10^5)$.

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 8. 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.

6. Fitness landscape analysis

Fitness landscape analysis has been performed on two separate problem instances of UBP$(6, 2, 3, 10^5)$ and UBP$(15, 5, 10, 10^5)$ instance generators. The GA_{Binary} variant has been taken as the base reference in this part of our studies. That is, the active entries of the policy matrix take values 1

Table 5: Performance comparison of heuristics 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 Lee and Lee (1985)	-	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_{Binary}	**99.99**	99.61	**98.42**	**99.58**	**99.55**	96.75	96.96	**98.45**	98.46	97.63
GA_{FFinit}	**99.99**	99.61	98.15	99.10	99.25	96.05	98.28	98.43	97.87	96.92

or 2. The experiment consists of evaluating the fitness value of all possible individuals for UBP$(6, 2, 3, 10^5)$ and a randomly sampled subset of all possible individuals of the UBP$(15, 5, 10, 10^5)$ instance generators. Considering the instances generated by UBP$(6, 2, 3, 10^5)$, the entire fitness landscape has been 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}, a full coverage of the landscape is intractable. This is due to the fact that some individuals represent low quality matrices and it takes a much longer time to evaluate them; poorer solutions tend to open many more bins, and processing these increases the processing time. Accordingly, a full coverage of the entire landscape has been avoided by sampling the landscape according to a uniform random distribution with a probability equal to 10^{-4}.

Moreover, redundant individuals of the instances generated by UBP$(15, 5, 10, 10^5)$ has been 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 in the same fitness value. The total number of possible individuals after redundancy elimination and prior to random sampling for UBP$(15, 5, 10, 10^5)$ is $78,129,765$. After all the pre-processing, the individuals are evaluated by the bin packing program.

The Fitness-Distance Correlation (FDC) as well as Correlation Length (CL) is then calculated for instances of each UBP. The FDC measure, proposed in Jones and Forrest (1995) , is a measure of search difficulty. Suppose that $F = \{f_1, f_2, \cdot f_n\}$ is a set of n individual fitnesses. Furthermore, suppose that $D = \{d_1, d_2, \cdot d_n\}$ 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 σ_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 $[-1, 1]$. A value closer to 1, indicates that the underlying problem is a misleading 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 (ℓ) 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 ℓ 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.

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

The plots in Fig.9(a) and Fig.9(c) demonstrates the fitness value for each individual in UBP$(6, 2, 3, 10^5)$ and UBP$(15, 5, 10, 10^5)$, respectively. The X axis represents the individuals and the Y axis the fitness value. This is while figures 9(b) and 9(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 their closest optima respectively. The X axis represents the hamming distance between individuals and the Y axis represents the fitness value. Moreover, figures 10(a) and 10(b) demonstrates the number of individuals with a specific hamming distance to the closest optimum for

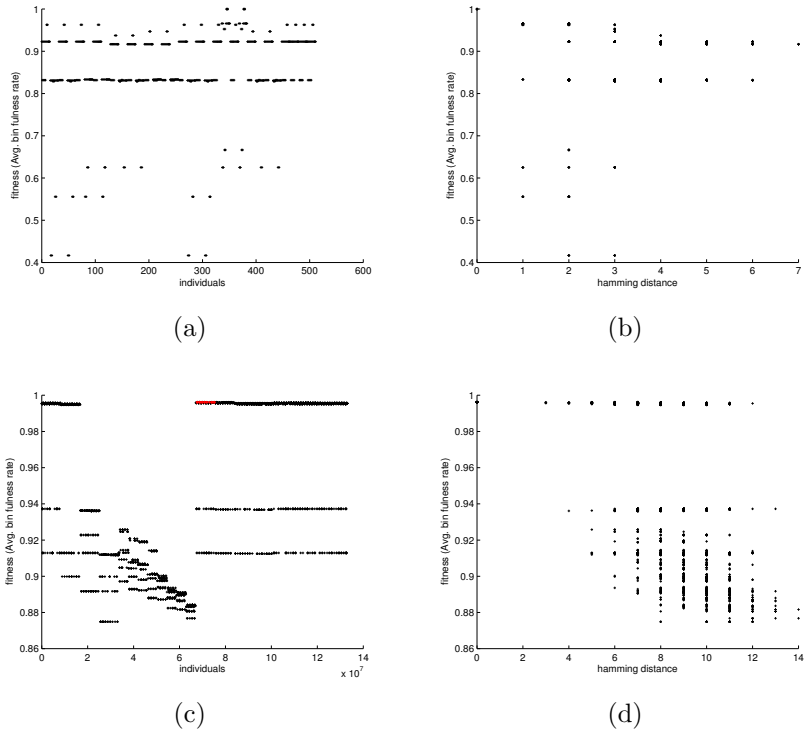
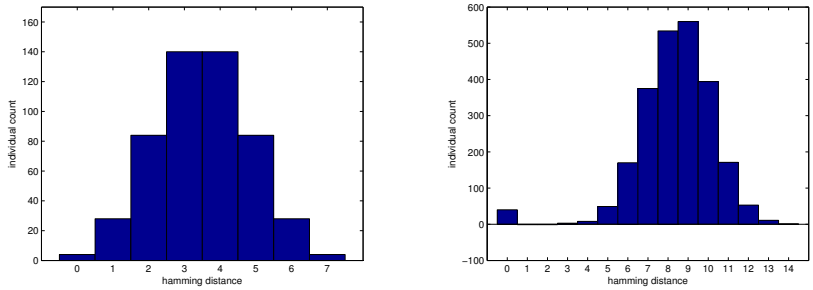


Figure 9: Fitness landscape analysis results for UBP$(6, 2, 3, 10^5)$ (a,b) and UBP$(15, 5, 10, 10^5)$ (c,d).

problems 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 solution and fitness for UBP$(6, 2, 3, 10^5)$, while there is a correlation for UBP$(15, 5, 10, 10^5)$.

7. 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



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

Figure 10: 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.

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.

The fitness landscape analysis of CHAMP-GA on some instances has confirmed that the search space of policies are sometimes rugged and sometimes they are not with neutral regions depending on the instance 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 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.

References

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

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.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R., Dec 2013. Hyper-heuristics. J Oper Res Soc 64 (12), 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. Vol. 146 of International Series in Operations Research and Management Science. Springer US, pp. 449–468.

Burke, E. K., Hyde, M. R., Kendall, G., September 2006. Evolving bin packing heuristics with genetic programming. In: Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006). Vol. 4193 of Lecture Notes in Computer Science. 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. Vol. 1 of Intelligent Systems Reference Library. Springer Berlin Heidelberg, 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. GECCO '07. ACM, New York, NY, USA, pp. 1559–1565.

Burke, E. K., Hyde, M. R., Kendall, G., Woodward, J. R., 25-28 Sep. 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., Srensen, K. (Eds.), Adaptive and Multilevel Metaheuristics. Vol. 136 of Studies in Computational Intelligence. Springer Berlin / Heidelberg, 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., Galambos, G., Martello, S., Vigo, D., 1999. Bin packing approximation algorithms: Combinatorial analysis. In: Du, D.-Z., Pardalos, P. (Eds.), Handbook of Combinatorial Optimization. Vol. 1 of Intelligent Systems Reference Library. Kluwer Academic Publishers, pp. 151–207.

Csirik, J., Woeginger, G., 1998. On-line packing and covering problems. In: Fiat, A., Woeginger, G. (Eds.), Online Algorithms. Vol. 1442 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 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., Mar. 2011. Multi-armed Bandit Allocation Indices, 2nd Edition. Wiley.

Gittins, J. C., 1979. Bandit processes and dynamic allocation indices. Journal of the Royal Statistical Society. Series B (Methodological) 41 (2), 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 (4), 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 (3), 562–572.

Martello, S., Toth, P., July 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 hyperheuristics. Intelligent Data Analysis 12 (1), 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. GECCO '11. ACM, New York, NY, USA, pp. 2011–2018.

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 (2), pp. 438–445.

Richey, M. B., 1991. Improved bounds for harmonic-based bin packing algorithms. Discrete Applied Mathematics 34 (1-3), 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, Ch. 17, pp. 529–556.

Ross, P., Schulenburg, S., Marn-Blzquez, 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., Jrgens, C., 1997. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers & Operations Research 24 (7), 627 – 645.

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

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. CEC'09. IEEE Press, Piscataway, NJ, USA, pp. 399–406.

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 (3), 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. Vol. 5199 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 1140–1149.

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