Comparing Two Models to Generate Hyper-heuristics for the 2D-Regular Bin-Packing Problem

H. Terashima-Marín, C. J. Farías Zárate
ITESM-Intelligent Systems
Av. E. Garza Sada 2501
Monterrey, NL, 64849 Mexico
terashima@itesm.mx
claudia.farias@ge.com

P. Ross
School of Computing
Napier University
Edinburgh EH10 5DT UK
P.Ross@napier.ac.uk

M. Valenzuela-Rendón
ITESM-Intelligent Systems
Av. E. Garza Sada 2501
Monterrey, NL, 64849 Mexico
valenzuela@itesm.mx

ABSTRACT

The idea behind hyper-heuristics is to discover some combination of straightforward heuristics to solve a wide range of problems. To be worthwhile, such combination should outperform the single heuristics. This paper presents two Evolutionary-Computation-based Models to produce hyper-heuristics that solve two-dimensional bin-packing problems. The first model uses an XCS-type Learning Classifier System which learns a solution procedure when solving individual problems. The second model is based on a GA that uses a variable-length representation, which evolves combinations of condition-action rules producing hyper-heuristics after going through a learning process which includes training and testing phases. Both approaches, when tested and compared using a large set of benchmark problems, perform better than the combinations of single heuristics. The testbed is composed of problems used in other similar studies in the literature. Some additional instances of the testbed were randomly generated.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods and Search* A category including the fourth, optional field follows...; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Algorithms

Keywords

Evolutionary Computation, Hyper-heuristics, Optimization, Cutting and Packing Problems

GECCO'07, July 7–11, 2007, London, England, United Kingdom

1. INTRODUCTION

Cutting stock and packing problems are widely studied because their many applications ranging from clothing and metal to engineering and shipbuilding. The problems belong to the class of most difficult problems known as NP-hard [8]. Given a set of pieces, the problem is to generate cutting patterns from sheets of stock material, or objects, that optimize certain objectives, such as to minimize the trim loss, or the number of objects used. In this particular investigation problems involve packing only 2D-rectangular pieces. Since many precise requirements and constraints vary from industry to industry, many different approaches and techniques have been proposed for solving the problem [13]. For small combinatorial problems, exact methods like linear programming can be applied. However, when larger and more complex problems appear, exact solutions are not a reasonable choice since the search space grows exponentially, and so does the time for finding the optimal solution. Various heuristic and approximate approaches have been proposed that guarantee finding near optimal solutions. However, it has not been possible to find a reliable method to solve all instances of a given problem. In general, some methods work well for particular instances, but not for all of them.

The aim of this paper is to compare two different alternatives on the usage of evolutionary approaches to generate hyper-heuristics when solving 2D-rectangular bin-packing problems. A hyper-heuristic is used to define a high-level heuristic that controls low-level heuristics [4]. The hyper-heuristic should decide when and where to apply each single low-level heuristic, depending on the given problem state. The choice of low-level heuristics may depend on the features of the problem state, such as CPU time, expected number of solutions, values on the objective function, etcetera. Selecting a particular heuristic is dynamic, and depends on both the problem state produced by the previous heuristic applied and the search space to be explored in that point of time. In recent work which is based on the research by Ross et al. [19] on unidimensional binpacking, evolutionary approaches have been used to generate hyper-heuristics for the 2D-Regular Cutting Stock Problems. Terashima et al. [22] use the XCS Classifier System to generate the hyper-heuristics. In other related work [21], authors use a Genetic Algorithm with variable-length representation to produce hyper-heuristics. Both previous approaches deliver very competitive results for a set of different problem instances, beating

in fact, results produced by the single heuristics. These methods assemble a combination of single heuristics (selection and placement), and this combination is formed taking into account the quality of partial solutions provided by the single heuristics.

Since both methods were tested individually, the investigation in this article focuses on refining and comparing them against each other when faced with the same testbed and under the same conditions. Results confirm the effectiveness of both approaches against the single heuristics.

The paper is organized as follows. Section 2 describes the 2D bin-packing problem. Section 3 presents the solution methods proposed and its justification. This is followed by the experimental setup, the results, their analysis and discussion in section 4. Finally, in section 5 we include our conclusions and some ideas for future work.

2. CUTTING AND PACKING PROBLEMS

The Cutting Stock Problem (CuSP) is among the earliest problems in the literature of operational research. In 1939, Kantorovich studied obvious applications in all the industries whose products were in a flat sheet form; this research was published in 1960 [15]. Since then, there have been many investigations on the problem: an abstract description of the different solution methods which have been given to the problem [11]; the applications and solutions to the CuSP problem [6]; and the solution methods of the problem [5]. Due to the diversity of problems and applications, Dyckhoff [6] has proposed a very complete and systematic categorization of cutting and packing problems. His survey integrates a general system of 96 problems for the Cutting Stock with four main features and their subtypes as follows: Dimensionality: One (1), Two (2), Three (3) or n; Assignation form (All the larger objects and a selection of small figures (B), or A selection of large objects and all the small figures (V)); Assortment of large objects (One object (O), Identical shapes (I), or Different Shapes (D)); and Assortment of small figures (Few figures of different shapes (F), Many figures of different shapes (M), Many figures of few of different and incongruent shapes (R), or Congruent shapes (C)). Then our work will be limited to a 2VIC-Cutting Stock Problem. According to a new and more complete typology recently proposed by Wäscher et al. [25] the problem falls into the categories of two-dimensional, input (value) minimization, weakly heterogeneous assortment of small items, identical large objects, and regular (rectangular) small items. The problem is considered as a Single Bin-Size 2D-Bin-Packing Problem.

3. SOLUTION APPROACHES

In the literature one can see that Evolutionary Computation has been used in few CuSP investigations. Recently, Hopper and Turton [13] have presented an empirical study on the usage of Meta-Heuristics for solving 2D Rectangular Bin Packing Problems. Evolutionary Computation usually includes several types of evolutionary algorithms [23]: Genetic Algorithms [9,12], Evolutionary Strategies [18,20], and Evolutionary Programming [1,7]. In this research we use two Evolutionary-Computation-based models to produce hyper-heuristics: an XCS-based approach; and a GA with variable length chromosomes, a resemblance of what is called a *messy*-GA [10].

3.1 The Set of Heuristics Used

In a one dimensional packing problem, the related heuristics refer to the way the pieces are selected and the bins in which they will be packed. For a two dimensional problem, additional difficulty is introduced by defining the exact location of the figures, that is, where a particular figure should be placed inside the object. In this investigation two kinds of heuristics were considered: for selecting the figures and objects, and for placing the figures into the objects. Some of the heuristics were taken from the literature, others were adapted, and some other variations developed by us. We chose the most representative heuristics in its type, considering their individual performance presented in related studies and also in an initial experimentation on a collection of benchmark problems. The selection heuristics used in this research are:

- First Fit (FF).- Consider the opened objects in increasing order and place the item in the first one where it fits.
- First Fit Decreasing (FFD).- Sort pieces in decreasing order, and the largest one is placed according to FF.
- Filler + FFD.- It places as many pieces as possible within the open objects. If at least one piece has been placed, the algorithm stops. The FFD algorithm is applied, otherwise.
- Next Fit (NF).- Use the current object to place the next piece, otherwise open a new one and place the piece there.
- Next Fit Decreasing (NFD).- Sort the pieces in decreasing order, and the largest one is placed according to NF.
- Best Fit (BF).- It places the item in the opened object where it best fits, that is, in the object that leaves minimum waste.
- Worst Fit (WF).- It places the item in the opened object where it worst fits (with the largest available room).
- Djang and Fitch (DJD).- It places items in an object, taking items by decreasing size, until the object is at least a third full. Then, it initializes w indicating the allowed waste, and looks for combinations of one, two, or three items producing a waste w. If any combination fails, it increases w accordingly. We adapted this heuristic to consider the initial filling different to a third, and the combinations for getting the allowed waste up to five items.

Some of these heuristics are described also in Ross et al. [19] and Hopper et al. [13].

The placement heuristics belong to the class of bottom-left heuristics, that is, they keep the bottom-left stability in the layout. They are based on a sliding technique. The placement heuristics we used are:

- Bottom-Left (BL) [14].- It starts at the upper corner of the object, then the piece slides vertically, all the way down, until it hits another piece, it continues sliding to the left (in straight line) as far as possible. A sequence of down and left movements is repeated until the piece reaches a stable position.

- Improved-Bottom Left (BLLT) [16].- It is similar to the above heuristic, but instead of moving the piece all the way and straight to the left, it keeps sliding it over the borderline of the bottom pieces until it reaches a stable position.

Both heuristics were modified to generate two new heuristics to consider rotation in the piece to place. These heuristics are called BLR and BLLTR.

There are 40 of those combinations involving selection and placement single heuristics and they are shown in Table 1.

Table 1: Representation of actions.

Action	Selection Heuristic	Placement Heuristic
1	First Fit (FF)	Bottom-Left (BL)
2		Bottom-Left Rotate (BLR)
3		Improved Bottom-Left(BLLT)
4		Improved Bottom-Left Rotate(BLLTR)
5	First Fit Decreasing (FFD)	Bottom-Left (BL)
6		Bottom-Left Rotate (BLR)
7		Improved Bottom-Left(BLLT)
8		Improved Bottom-Left Rotate(BLLTR)
9	First Fit Increasing (FFI)	Bottom-Left (BL)
10		Bottom-Left Rotate (BLR)
11		Improved Bottom-Left(BLLT)
12		Improved Bottom-Left Rotate(BLLTR)
13	Filler+FFD	Bottom-Left (BL)
14		Bottom-Left Rotate (BLR)
15		Improved Bottom-Left(BLLT)
16		Improved Bottom-Left Rotate(BLLTR)
17	Next Fit (NF)	Bottom-Left (BL)
18		Bottom-Left Rotate (BLR)
19		Improved Bottom-Left(BLLT)
20		Improved Bottom-Left Rotate(BLLTR)
21	Next Fit Decreasing (NFD)	Bottom-Left (BL)
22		Bottom-Left Rotate (BLR)
23		Improved Bottom-Left(BLLT)
24		Improved Bottom-Left Rotate(BLLTR)
25	Best Fit (BF)	Bottom-Left (BL)
26		Bottom-Left Rotate (BLR)
27		Improved Bottom-Left(BLLT)
28		Improved Bottom-Left Rotate(BLLTR)
29	Best Fit Decreasing (BFD)	Bottom-Left (BL)
30		Bottom-Left Rotate (BLR)
31		Improved Bottom-Left(BLLT)
32		Improved Bottom-Left Rotate(BLLTR)
33	Worst Fit (WF)	Bottom-Left (BL)
34		Bottom-Left Rotate (BLR)
35		Improved Bottom-Left(BLLT)
36		Improved Bottom-Left Rotate(BLLTR)
37	Djang and Finch (DJD)	Bottom-Left (BL)
38		Bottom-Left Rotate (BLR)
39		Improved Bottom-Left(BLLT)
40		Improved Bottom-Left Rotate(BLLTR)

The concept of hyper-heuristic is motivated by the objective to provide a more general procedure for optimization [4]. Meta-heuristics methods usually solve problems by operating directly on the problem. Hyper-heuristics deal with the process to choose the right heuristic for solving the problem at hand. The idea is to discover a combination of simple heuristics that can perform well on a whole range of problems. For real applications, exhaustive methods are not a practical approach. It is common to sacrifice quality of solutions by using quick and simple heuristics to solve problems. Many heuristics have been developed for specific problems. But, is there a single heuristic for a problem that solves all instances well? The immediate answer is no. Certain problems may contain features that would make specific heuristic to work well, but those features may not be suitable for other heuristics. The idea with hyper-heuristics is to combine heuristics in such a way that a heuristic's strengths make up for the drawbacks of another.

3.2 Model 1: XCS-based Approach

Classifier Systems (of the Michigan type) evolve a set of condition-action rules or 'classifiers' and periodically use a Genetic Algorithms with the ordinary genetic operators such as selection, crossover and mutation, to breed new rules from old ones. What is obtained is a set of rules representing an adaptive system, that given a change in the environment, would react accordingly. The LCS interacts with the environment perceiving situations σ, usually coded as binary strings of length L, performing actions α, and finding scalar feedback ρ. Knowledge is represented in a population $[P]$ of classifiers. The population size is given by the parameter N. The system interacts with the environment via detectors (input) and effectors (actions). The environment also provides a scalar reinforcement, also called reward. The module [P] represents a population of classifiers where the left side consists of the conditions, and the right side indicates the environmental actions. Given an input a *match set* [M] is formed by those classifiers in [P] that match their conditions with the given situation in the environment. The system then computes a prediction $P(a_i)$ for each action represented in [M]. Actions are selected from [M] to form an *action set* [A]. Several action-selection methods have been studied in the literature. One action is sent to the effectors and an immediate reward may be returned by the environment. The most important elements in a classifier system are the Reinforcement component, the Discovery component, and the Fitness calculation scheme. The Reinforcement component consists of updating the ρ, ϵ (error on the prediction parameter), and F (classifier's fitness) parameters of classifiers in the previous action set $[A]_{-1}$. The discovery component is based on a Genetic Algorithm working in the match set [M] to generate new classifiers. The fitness calculation scheme provides the quantity to update the classifier's fitness depending on the classifier's accuracy relative to the accuracies of the other classifiers in the set.

The above two concepts were merged to solve 2D bin-packing problems, following previous work by Ross et al. [19] for one dimensional bin packing. An XCS type classifier system [24] was used to form the hyper-heuristics. The block diagram of the system is shown in Figure 1. The XCS evolves a behavior model which determines the possible actions for all situations or states of the problem. In this particular model, the actions are given by the selection and placement heuristics to be applied in a given situation. To find the appropriate set of rules linking problem states with heuristics, the environment was coded using particular features in the problem at hand. For example, the environment informs the classifier system the size of the objects, the amount and ratio of figures to be packed. Then, each classifier associates the problem state with a selection and placement heuristic, which are applied until certain condition is met. The process continues until the problem is completely solved.

3.2.1 Representation

A rule determines the relationship between a condition and an action.

The condition segment in a classifier has the following information:

- **Height representation:** SH.- Items up to 1/3 of object height; MH.- Items from 1/3 up to 1/2 of object height; and LH.- Items over 1/2 of object height
- **Width representation:** SW.- Items up to 1/3 of object width; MW.- Items from 1/3 up to 1/2 of object width; and LW.- Items over 1/2 of object width
- **Area representation:** SA.- Items up to 1/4 of object area; MA.- Items from 1/4 up to 1/3 of object area;

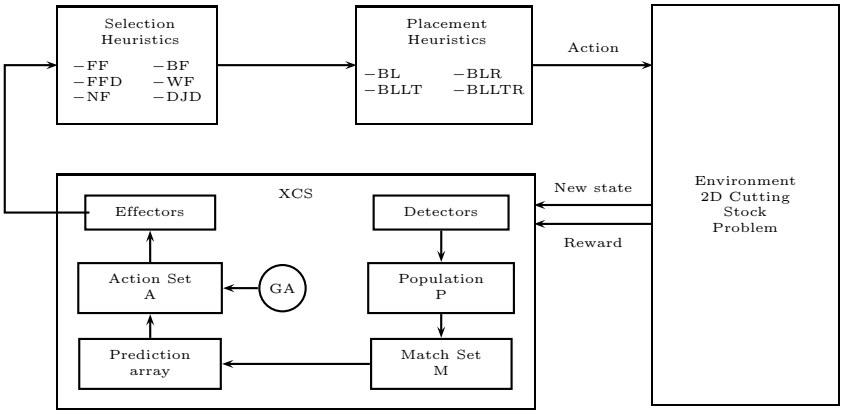

Figure 1: Model combining Hyper-Heuristics and the XCS Classifier System.

LA.- Items from 1/3 up to 1/2 of object area; HA.- Items over 1/2 of object area

- **R.-** Ratio of items left to be cut

The action segment contains the following information: SCH, Selection Heuristic and PMH, Placement Heuristic.

Each part was coded into categories according to dimensions in the height, width, and area of objects (small, medium, and large, and we added huge for area). Each of them has a proportion of items (00: 0-10%; 01: 10-20%; 10: 20-50%; and 11: 50-100%). The percentages of items remaining to be cut are as follows: 000: 0-14%; 001: 14-28%; 011: 28-42%; 010: 42-56%; 110: 56-70%; 100: 70-84%; and 111: 84-100%.

The action was selected from all possible combinations of selection and placement heuristics, considering also the possibility of rotating an object by 90 degrees. Those combinations were previously shown in Table 1.

The general procedure of the method has the following steps:

1. The XCS generates a random population of classifiers.
2. The current problem state is matched against those rules.
3. With the Matching Set and the Prediction Array an Action Set is formed from which the best classifier is taken to perform the indicated action (a combination of selection and placement heuristic). That action is carried out until an object is completely full or no other remaining piece fits in that object.
4. Reward is applied depending on the selected reinforcement scheme (Single or Multi-Step). In the single-step, reward is paid after every application of the selected combination of heuristics, whereas in the multi-step, the reward is updated after a complete solution is delivered.
5. Once an instance has been completely solved, the best classifiers are kept in the population and the solution process starts again for that instance.
6. The procedure continues until a pre-established number of cycles is reached.

3.2.2 *Fitness Function*

For measuring the quality of solutions produced by Model 1, the following equation was used:

$$FF = \frac{\sum_{u=1}^{No} P_u^2}{No} \quad (1)$$

where No is the total number of objects used, and P_u is the percentage of utilization for each object u, given by $P_u = \frac{\sum_{j=1}^{n} Ap_j}{Ao}$, where Ap represents the item area; Ao the object area; and n the number of items inside the object.

3.3 Model 2: GA-Based Approach

In this model, a GA with variable-length individuals is proposed to find a combination of single heuristics (selection and placement) to solve efficiently a wide variety of instances of 2D-Regular bin-packing problems. The basic idea is that, given a problem state P, this is associated with the closest point in the chromosome which carries the selection and placement to be applied. This application will transform the problem to a new state P'. The purpose is to solve a problem by constructing the answer, deciding on the heuristic to apply at each step. The current state P of the problem is a much-simplified representation of the actual state, and is described in more detail in section 3.3.1. A chromosome represents a set of labelled points within this simplified problem state space; the label of any point is a heuristic. The chromosome therefore represents a complete recipe for solving a problem, using a simple algorithm: until the problem is solved, (a) determine the current problem state P, (b) find the nearest point to it, (c) apply the heuristic attached to the point, and (d) update the state. The GA is looking for the chromosome (a hyper-heuristic) which contains the rules that apply best to any intermediate state in the solving process of a given instance. The instances are divided into two groups: the training and testing sets. The general procedure consists in solving first all instances in both sets with the single heuristics (each is a combination of a selection and a placement heuristic). The best solution for each instance is kept, that is later used also by the GA proposed. Then the GA works on the training set until termination criterion is met and a general hyper-heuristic is produced. All instances in both the testing and training sets are then solved with this general hyper-heuristic. The complete process is presented in Figure 2.

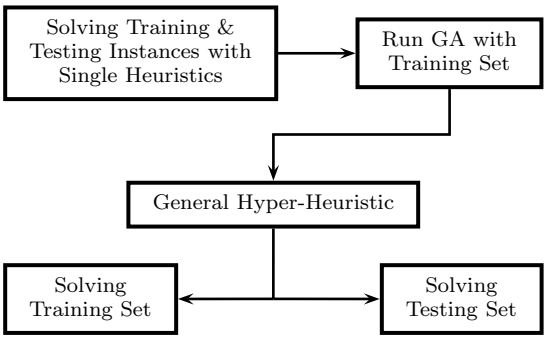

Figure 2: Solution Model.

3.3.1 Representation

Each chromosome is composed of a series of *blocks*. Each block j includes nine numbers. The first eight represent an instance of the problem state. The initial four numbers indicate the percentage of pieces that remain to be packed according to the following categories (Ao is the object area, Ap is the item area): h_j, huge items ($Ao/2 < Ap$), l_j large items ($Ao/3 < Ap \leq Ao/2$), m_j medium items ($Ao/4 < Ap \leq Ao/3$), and s_j, small items ($Ap \leq Ao/4$); the following three numbers indicate the percentage of items that remain to be packed according to a categorization based on the height (Ho is the object height, Hp is the item height): l_j large items ($Ho/2 < Hp$), m_j medium items ($Ho/3 < Hp \leq Ho/2$), and s_j, small items ($Hp \leq Ho/3$): The eighth number, r_j, represents the percentage of the total items that remain to be packed. The ninth number is an integer indicating the combination of heuristics (selection and placement), associated with this instance, see Table 1.

For a given problem state, the first eight numbers lie in the range 0 to 1, so a block can be regarded as a labelled point in the eight-dimensional unit cube. The label is the ninth number, which identifies a selection and a placement heuristic. The GA's task is to position an unspecified number of such labelled points, and the problem-solving algorithm is simply this: given a point representing the problem's current state, find the nearest labelled point and do what the label says, and repeat until all items in the problem have been selected and placed. To determine 'nearest' we use Euclidean distance. If we only allowed labelled points to lie inside the unit cube, there would be possible edge effects because the nearest labelled point to a corner would 'cover' more volume than interior points. To counteract this, we permit labelled points to lie anywhere in the $(-3, 3)^8$ cube, although the problem state can still only be in the unit cube. Experience suggests that the slight asymmetry of this does not matter.

The action was selected from all possible combinations of selection and placement heuristics, taking also into consideration the possibility of rotating the items.

3.3.2 Genetic Operators

We dealt in this investigation with two crossover and three mutation operators. The first crossover operator is very similar to the normal two-point crossover. Since the number of blocks in each chromosome is variable, each parent selects the first and last block independently. However the points selected inside each corresponding block are the same for both parents. The blocks and poinst are chosen using a uniform distribution. The other crossover operator works at block level, and it is very similar to the normal one-point crossover. This operator exchanges 10% of blocks between parents, meaning that the first child obtains 90% of information form the first parent, and 10% from the second one.

The first mutation operator randomly generates a new block and adds it at the end of the chromosome; the second operator randomly selects and eliminates a block within the chromosome; and the last one randomly selects a block in the chromosome and a position inside that block to replace it with a new number between -3 and 3, generated with a normal distribution with mean 0.5 and truncated accordingly.

3.3.3 The Fitness Function

The quality of solution produced by either a single heuristic or a hyper-heuristic for a given instance, is given by equation 1, also used in Model 1. Now, how is the fitness of a chromosome measured during the GA cycle? To do this, first, it is necessary to compute the fitness produced by each individual combination of selection and placement heuristics, for each instance. The best heuristic combination and its result, for each specified instance i are stored (let us call it BSH_i). These results are prepared in advance of running the GA.

The GA cycle consists of the following steps:

1. Generate initial population
2. Assign 5 problems to each chromosome and get its fitness
3. Apply selection (tournament), crossover and mutation operators to produce two children
4. Assign 5 problems to each new child and get its fitness
5. Replace the two worst individulas with the new offspring
6. Assign a new problem to every individual in the new population and recompute fitness
7. Repeat from step 3 until a termination criterion is reached.

To compute the fitness for each chromosome (at steps 2 and 4 of the above cylce), the distance between the solution obtained by that individual with respect to the best result given by the single heuristic (BSH_i) is measured. The fitness is a weighted average and it is given by:

$$FF(HH) = \frac{\sum_{k=1}^{5} P_k \cdot (FF_k - BSH_k)}{\sum_{k=1}^{5} P_k} \qquad (2)$$

where P_k is the number of times the k-th assigned instance has been tackled so far; BSH_k is the best fitness obtained for the k-th assigned instance by a single heuristic; and FF_k is the fitness obtained by the hyper-heuristic for the k-th assigned instance (using equation 1).

After each generation l, a new problem is assigned to each individual m in the population and its fitness is recomputed by a weighted average as follows:

$$FF_m^l = \frac{FF_m^{l-1} \cdot mp_m + FF(m)}{mp + 1} \qquad (3)$$

where FF_m^{l-1} is the fitness for individual m in the previous generation; mp_m is the number of problems individual m has seen so far; $FF(m)$ is the fitness obtained by individual m for the new problem and computed with equation 1.

3.3.4 GA Parameter Set

After previous experimentation, the parameters for the GA used in this investigation were set as follows: population size, 50; number of generations, 400; crossover probability, 1.0; and mutation probability, 0.1.

4. EXPERIMENTS AND RESULTS

This section presents the experiments carried out during the investigation and the results obtained. Problems from several sources have been used in this research. Part of the benchmark set was taken from the literature (the OR-Library [2], set by Martello and Vigo [17], set by Berkey and Wang [3], set by Terashima et al. [22]), and the rest is composed of randomly generated instances for which an optimal solution is known. The collection includes 1080 different instances. They were divided into two groups (A and B) of 540 instances each (chosen at random). Aiming at testing the model effectiveness, three kinds of experiments were carried out.

4.1 Experimenting with Single Heuristics

All combinations of single heuristics were tested with the two instance groups. The intention for doing this was to establish the basis for comparison on the performance of the hyper-heuristics delivered by the two proposed models. The results of the single heuristics in Group A are shown in Table 2. Table 3 shows results when Single Heuristics were tested with Group B.

Results are grouped in sets of four heuristics in which the selection heuristic is common in a given set. For example, column 2 of Table 2 indicates results for the first four combinations in which the common selection heuristic is First Fit. Results are compared against those generated by the best single heuristics. Figures in cells indicate the percentage of problems solved with a particular number of extra objects (left column) when compared with results provided by the best heuristic. For example, on the column labeled 'FFD', heuristics 5 to 8 solved 70.96% (this is the average performance on the four heuristics) of the instances in set A with the same number of objects as the best heuristic for each given instance. In both Tables (2 and 3) best results are provided by the set whose common selection heuristic is the DJD (figures in bold). This is somehow as expected since the DJD heuristic is, in general, a very good heuristic for Bin-packing. There are other sets of heuristics showing also a very good performance (13-16, 29-32) in which the selection heuristic is the Filler plus First Fit, and Best Fit Decreasing, respectively.

4.2 Experimenting with Model 1

In previous published work [22], results of this model were compared against other conventional approaches for a limited set of instances. In this experiment, the idea is to compare the results of Model 1 not only against the combinations of single heuristics, but also against another model producing hyper-heuristics using a different scheme (Model 2). The instance set used in this case is also much larger, so the performance by each approach could be appreciated in more detail. Table 4 shows the outcome for the XCS-based model using both rewarding schemes (Single-Step and Multiple-Step) for both instance sets. The way of rewarding classifiers does not seem to make any difference in performance. Figures in cells represent the percentage of extra objects used compared against the result provided by the best heuristic for each particular instance problem in each set. In group A, both columns show practically the same results, however, for set B, there is just a slight improvement when using Multiple-Step. It is interesting to observe in the results, that the XCS-type solves around 4% of problems in set A with one less object than the best single heuristics, and around 10% of problems in set B.

4.3 Experimenting with Model 2

To produce the hyper-heuristics, Model 2 was tested as follows. First, the number of complete runs was established to five. Then for each particular run, and after the training phase was finished (held with set A only), the best five hyper-heuristics were taken, tested and compared with the two instance sets (separately), and then the best one was chosen for the general comparison (reported in Table 6). For instance, Run 1 produced results shown in Table 5. The left segment shows results with the best five hyper-heuristics for the training set A, and the right part shows results for the testing set B. That means, for example, that **HHI-1A** is the hyper-heuristic obtained with set A, and the corresponding results when that hyper-heuristic was tested with set B are shown in column labeled **HHI-1B**. The best five hyper-heuristics produced by that run solve on average 88.6% (1.61 + 94.09 + 1.61 + 89.78 + 1.08 + 92.47 +0 + 81.18 +0 + 81.18 divided by 5) of problems in set A with at least the same number of objects as the best single heuristic. Performance on set B is quite similar, confirming the effectiveness of this model. Table 6 summarizes results for all five runs. The best hyper-heuristic for each run was selected and kept in the left columns (second to sixth). The corresponding results for those hyper-heuristics on set B are presented in the right part of the Table (columns seventh to eleventh). The average performance of these heuristics on set A is around 89.46% considering results with 0 or -1 extra objects. The average performance of the same heuristics, but for set B, lies around 85.6%. It is interesting to observe in these results that, for set B, there are some hyper-heuristics that obtain a percentage of problems for which the number of objects is two less than the one obtained by the best single heuristics.

4.4 Comparison and Summary of Results

Table 7 summarizes results when comparing the performance of single heuristics against both models used to generate hyper-heuristics. The second and third columns include results provided by the best single heuristics for sets A and B, and shown in bold in Tables 2 and 3. Those results are provided by combinations of single heuristics having in common the DJD selection heuristic. The fourth and fifth columns show the best results provided by the XCS-based model which are produced when the Multiple-Step rewarding scheme is used. The last two columns are the results for the best hyper-heuristic produced for all runs by Model 2. It is clear that the two models used to generate hyper-heuristics outperform the single heuristics. Model 2 is, on average for both groups, slightly better than Model 1.

Table 2: Single Heuristics: Number of extra objects for problems in Set A.

	FF	FFD	FFI	Filler	NF	NFD	BF	BFD	WF	DJD
Obj	**1-4**	**5-8**	**9-12**	**13-16**	**17-20**	**21-24**	**25-28**	**29-32**	**33-36**	**37-40**
0	30.65	70.43	18.55	75.54	30.91	64.65	31.18	70.43	30.65	**76.61**
1	31.99	27.69	21.24	22.58	30.11	24.6	31.18	27.15	31.99	**21.51**
2	21.24	1.61	8.33	1.61	22.04	0.81	21.77	2.15	21.24	**1.61**
3	8.87	0.27	7.53	0.27	9.68	0.13	8.6	0.27	8.87	**0.27**
4	5.65		9.14		5.38		5.38		5.65	
> 4	1.61		35.22		1.88		1.88		1.61	

Table 3: Single Heuristics: Number of Extra Objects for Problems in Set B.

	FF	FFD	FFI	Filler	NF	NFD	BF	BFD	WF	DJD
Obj	**1-4**	**5-8**	**9-12**	**13-16**	**17-20**	**21-24**	**25-28**	**29-32**	**33-36**	**37-40**
0	28.49	65.32	13.44	68.28	29.84	63.84	29.3	66.67	28.76	**71.77**
1	32.8	31.45	21.77	28.49	30.91	25.13	31.18	30.11	32.53	**25.27**
2	20.97	2.96	8.6	2.96	19.35	2.02	20.7	2.96	20.96	**2.69**
3	9.68	0	11.02	0	11.56	0	10.57	0	9.41	**0**
4	6.72	0.27	8.6	0.27	6.99	0.13	6.72	0.27	6.99	**0.27**
> 4	1.34		36.56		1.34		1.34		1.34	

Table 4: Model 1: Results obtained with Single-Step (SS) and Multiple-Step (MS) XCS for both problem sets.

Obj	XCS-SS-A	XCS-MS-A	XCS-SS-B	XCS-MS-B
-2	0	0	0	0
-1	4.07	4.07	11.5	10.4
0	85.56	85.60	70.6	75.4
1	10.37	10.33	17.9	14.2

Table 5: Model 2: Results on Run 1 with training and testing sets A and B.

Obj	HHI-1A	HHI-2A	HHI-3A	HHI-4A	HHI-5A	HHI-1B	HHI-2B	HHI-3B	HHI-4B	HHI-5B
-1	1.61	1.61	1.08	0	0	2.15	2.15	2.69	1.08	1.61
0	94.09	89.78	92.47	81.18	81.18	81.18	76.88	80.11	71.51	73.12
1	4.3	8.6	6.45	18.82	18.82	16.67	20.97	17.2	27.42	25.27
> 2	0	0	0	0	0	0	0	0	0	0

Table 6: Model 2: The best five hyper-heuristics for all runs and test sets A and B.

Obj	HHI-A	HHII-A	HHIII-A	HHIV-A	HHV-A	HHI-B	HHII-B	HHIII-B	HHIV-B	HHV-B
-2	0	0	0	0	0	0	0.54	0.54	1.08	0.54
-1	1.61	3.23	5.38	6.45	3.76	2.69	8.6	8.6	8.06	10.22
0	94.09	76.88	83.33	84.95	87.63	80.11	72.58	76.88	76.88	80.65
1	4.3	19.89	11.29	8.6	8.6	17.2	18.28	13.98	13.98	8.6

Table 7: Best results for Single Heuristics, Model 1 and Model 2.

Obj	SH-A	SH-B	XCS-A	XCS-B	HH-A	HH-B
-2						
-1			4.07	10.4	1.61	2.15
0	76.61	71.77	85.6	75.4	94.9	81.8
1	21.51	25.27	10.33	14.2	4.3	16.67
> 1	1.88	2.96				

5. CONCLUSIONS AND FUTURE WORK

This document has described experimental results for two models to generate hyper-heuristics for the 2D-Regular bin-packing problem. The first model uses a classifier system to form hyper-heuristics. The rules in the classifier system are composed of condition-action in which the condition is a representation of the problem state and the action indicates the combination of single heuristics (selection and placement) which should be applied. The second model is based on a variable-length GA which evolves combinations of condition-action rules representing problem states and associated selection and placement heuristics. Overall, both schemes identify efficiently hyper-heuristics that, when tested and compared with a large set of problem instances, show very competitive performance, in fact, much better than the best single heuristic for each instance. Of course, a hyper-heuristics approach has a major benefit compared to many other techniques. The significant search effort is applied to find a search algorithm; in this paper, the discovered algorithm is controlled either by a set of labelled points (in the messy-GA approach) or a set of rules (in the XCS approach). But once the algorithm has been found, it is extremely cheap and fast to apply to any new problem instance and does not involve any significant search effort.

Ideas for future work involve extending the proposed strategy to solve problems including other kinds of pieces such as polygonal, irregular, etcetera. It would be also interesting to work the approaches for other multidimensional problems.

6. ACKNOWLEDGMENTS

This research was supported by ITESM under the Research Chair CAT-010 and the CONACYT Project 41515.

7. REFERENCES

[1] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic programming: An Introduction*. Morgan Kaufmann Publishers, Inc, London, 1998.

[2] J. E. Beasley. Beasley operations research library. *Collection of problems for 2D packing and cutting*, 2003.

[3] J. O. Berkey and P. Y. Wang. Two-dimensional finite bin packing algorithms. *Journal of Operational Research Society*, 38:423–429, 1987.

[4] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern research technolology. In *Handbook of Metaheuristics*, pages 457–474. Kluwer Academic Publishers, 2003.

[5] C. H. Cheng, B. R. Fiering, and T. C. Chang. The cutting stock problem. a survey. *International Journal of Production Economics*, 36:291–305, 1994.

[6] H. Dyckhoff. A topology of cutting and packing problems. *European Journal of Operation Research*, 44:145–159, 1990.

[7] D. B. Fogel, L. A. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

[8] M. Garey and D. Johnson. *Computers and Intractability*. W.H. Freeman and Company, New York, 1979.

[9] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Adison Wesley, 1989.

[10] D. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, pages pp. 93–130, 1989.

[11] B. L. Golden. Approaches to the cutting stock problem. *AIIE Transactions*, 8:256–274, 1976.

[12] J. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[13] E. Hopper and B. C. Turton. An empirical study of meta-heuristics applied to 2D rectangular bin packing. *Studia Informatica Universalis*, pages 77–106, 2001.

[14] S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operations Research*, 88:165–181, 1966.

[15] L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6:366–422, 1960.

[16] D. Liu and H. Teng. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangle. *European Journal of Operations Research*, 112:413–419, 1999.

[17] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Dipartimento di Elettronica, Informatica e Sistematica*, 1998.

[18] I. Rechenberg. *Evolutionstrategie: Optimierung technischer systeme nach prinzipien dier biolischen evolution*. Frommann-Holzboog, Stuttgart, 1973.

[19] P. Ross, S. Schulenburg, J. M. Blázquez, and E. Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. *Proceedings of GECCO 2002*, pages 942–948, 2002.

[20] H. P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, Chinchester, UK, 1981.

[21] H. Terashima-Marín, C. J. Farías-Zárate, P. Ross, and M. Valenzuela-Rendón. A GA-based method to produce generalized hyper-heuristics for the 2D-regular cutting stock problem. *Proceedings of the Genetic and Evolutionary Computation Conference 2006*, pages 591–598, 2006.

[22] H. Terashima-Marín, E. J. Flores-Álvarez, and P. Ross. Hyper-heuristics and classifier systems for solving 2D-regular cutting stock problems. *Proceedings of the Genetic and Evolutionary Computation Conference 2005*, pages 637–643, 2005.

[23] R. A. Wilson and F. C. Keil. *The MIT Encyclopedia of the Cognitive Science*. MIT Press, Cambridge, Massachussets, 1999.

[24] S. W. Wilson. Generalization in the XCS classifier system. *Evolutionary Computation*, 1998.

[25] G. Wäscher, H. Haussner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operation Research*, 171, 2006.