A GA-Based Method to Produce Generalized Hyper-heuristics for the 2D-Regular Cutting Stock Problem

H. Terashima-Marín, C. J. Farías Zárate
ITESM-Intelligent Systems
Av. E. Garza Sada 2501
Monterrey, NL, 64849 Mexico
{terashima, a00789069}@itesm.mx

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 a GA-based method that produces general hyper-heuristics that solve two-dimensional cutting stock problems. The GA 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. Such hyper-heuristics, when tested with a large set of benchmark problems, produce outstanding results (optimal and near-optimal) for most of the cases. The testebed 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 Soving, Control Methods and Search*

General Terms

Algorithms

Keywords

Evolutionary Computation, Hyper-heuristics, Optimization, Cutting Stock

1. INTRODUCTION

Cutting stock is a problem widely studied because it has many applications ranging from clothing and metal to engineering and shipbuilding. The problem belongs 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 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 [14]. 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 explore a novel alternative on the usage of evolutionary approaches to generate hyper-heuristics when solving 2D-rectangular cutting stock 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. [21] on unidimensional binpacking, evolutionary approaches have been used to generate hyper-heuristics for the 2D-Regular Cutting Stock Problems. Terashima et al. [23] use the XCS Classifier System to generate the hyper-heuristics. In other related work [24], authors use a Genetic Algorithm with integer and fixed-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.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.
Copyright 2006 ACM 1-59593-186-4/06/0007 ...$5.00.

The investigation in this article, presents a method to generate a general hyper-heuristic intended to solve a wide variety of instances of 2D-regular cutting stock problems. The procedure learns the hyper-heuristic by going through a training phase using instances with a variety of features. The generated hyper-heuristic is tested later with a collection of unseen examples providing excellent results. The general method is based on a variable-length Genetic Algorithm, where the chromosome is conformed of a series of blocks, representing condition-action rules.

The paper is organized as follows. Section 2 describes the cutting stock problem. Section 3 presents the solution method 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. THE CUTTING STOCK PROBLEM

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 [16]. Since then, there have been many investigations on the problem, references of which are in different surveys that describe the CuSP's development and applications, from several points of view: an abstract description of the different solution methods which have been given to the problem [11]; the evolution of the problem with the objective of maximal production [12]; the applications and solutions to the CuSP problem [6]; and the solution methods of the problem [5].

Given a set $L = (a_1, a_2, ...a_n)$ of items to be cut, each one of size $s(a_i)\epsilon(0, Ao]$, from a set of cutting stock sheets (objects) of size Ao, the problem is to find cutting patterns, in such a way that the solution minimizes the number of used objects and the trim loss. This NP-problem, can be complicated depending very much on the number of variables, such as the number of figures, their shapes, the rotation angles, the maximum number of pieces to cut in an object, number of dimensions, and color, for example. 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:

1. Dimensionality: One (1), Two (2), Three (3) or n
2. Assignation form:
   (a) All the larger objects and a selection of small figures (B)
   (b) A selection of large objects and all the small figures (V)
3. Assortment of large objects:
   (a) One object (O)
   (b) Identical shapes (I)
   (c) Different Shapes (D)
4. Assortment of small figures:
   (a) Few figures of different shapes (F)
   (b) Many figures of different shapes (M)
   (c) Many figures of few of different and incongruent shapes (R)
   (d) Congruent shapes (C)

The extension of the CuSP and the objectives of this investigation restricted the problem to a Cutting Stock Problem of two dimensions (2). The dimensionality refers to the cutting action, as the cut will be done in both directions of length and width in the material. It is assumed that there are always enough resources to satisfy the demand, and there will be a total of requested figures cut in a stock material (V). The stock material will have identical shapes (I); and the experimentation will be done for rectangular shapes (C). Then our work will be limited to a 2VIC-Cutting Stock Problem.

3. SOLUTION APPROACH

In the literature one can see that Evolutionary Computation has been used in few CuSP investigations. Recently, Hopper and Turton [14] 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 [25]: Genetic Algorithms [9,13], Evolutionary Strategies [19,22], and Evolutionary Programming [1, 7]. In this research we use 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 such as the 2VIC-CuSP, 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:

- Next Fit (NF).- Use the current object to place the next piece, otherwise open a new one and place the piece there.
- First Fit (FF).- Consider the opened objects in increasing order and place the item in the first one where it fits.
- 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).
- Almost Worst Fit (AWF).- It places the item in the opened object with the second largest available room.

- First Fit Decreasing (FFD).- Sort the pieces in decreasing order, and the largest one is placed according to FF.
- Next Fit Decreasing (NFD).- Sort the pieces in decreasing order, and the largest one is placed according to NF.
- 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. [21] and Hopper et al. [14].

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) [15].- 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) [17] .- 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.

3.2 Combining Heuristics with the proposed GA

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. The search space might be too large, or the number and types of constraints may generate a complex space of feasible solutions. 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. The solution model used in this investigation carries features from previous work by Ross et al. [20], in which the main focus is to solve uni-dimensional bin-packing problems. In the research in this article, 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 cutting stock 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 3.2.1. A chromosome in the messy GA 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 heursitic attached to the point, and (d) update the state. The GA is looking for the chromosome (representing 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 the testing set. 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). This is carried out to keep the best solution that is later used also by the GA we propose. The next step is to let the GA work 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 1.

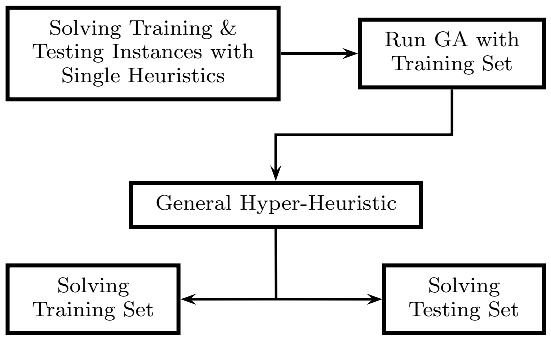

Figure 1: Solution Model.

3.2.1 Representation

Each chromosome is composed of a series of *blocks*. Each block j includes six numbers. The first five 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 fifth number, r_j, represents the percentage of the total items that remain to be packed. The sixth number is an integer indicating the combination of heuristics (selection and placement), associated with this instance.

For a given problem state, the inital five numbers would lie in a range between 0 and 1, so that the actual problem state is a point inside the unit five-dimensional space. Nevertheless, we allow the points defined in each block to lie outside the unit cube, so we redefined the range to be from -3 to 3. At each step, the algorithm applies that heuristic that is associated to the block that is closest to actual problem state. We measure the distance d between the problem state P' and the instance inside each block j using the following formula:

$$d^2=(h_j-h')^2+(l_j-l')^2+(m_j-m')^2+(s_j-s')^2+(r_j-r')^2 \quad (1)$$

where each term indicates the square of the difference for each category previously defined.

The action was selected from all possible combinations of selection and placement heuristics, taking also into consideration the possibility of rotating the items. There are 40 of those combinations 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)

3.2.2 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.

3.2.3 The Fitness Function

The quality of solution produced by either a single heuristic or a hyper-heuristic for a given instance, is based on the percentage of usage for each object 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. Then, the quality of each solution is given by

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

where No is the total number of objects used, and P_u is the percentage of utilization for each object u.

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:

$$FF(HH) = \frac{\sum_{k=1}^{5} P_k \cdot (FF_k - BSH_k)}{\sum_{k=1}^{5} P_k} \quad (3)$$

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 2).

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:

$$FF_m^l = \frac{FF_m^{l-1} \cdot mp_m + FF(m)}{mp + 1} \quad (4)$$

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 2.

3.2.4 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 [18], set by Berkey and Wang [3], set by Terashima et al. [23]), and the rest is composed of randomly generated instances for which an optimal solution is known. The collection includes 540 different instances. They were divided into two groups (A and B) of 270 instances each (chosen at random). Aiming at testing the model effectiveness, three kinds of experiments were carried out.

4.1 Experiment Type I

In this experiment, the instance set A is used as the training set. After this phase is complete, a general hyper-heuristic is produced by the GA, which is used in the testing phase to solve all instances in both sets A and B. Table 2 shows the hyper-heuristic obtained. We call it HH-I. It includes 9 rules indicating the different problem states and the associated action or heuristic to be applied. In the resulting hyper-heuristic, we can observe that despite the same single heuristic appears more than once in the action part, the conditions for applying it are quite different.

Table 2: Experiment Type I: Hyper-heuristic (HH-I) produced by the GA With group A as training set.

Huge	Large	Medium	Small	Remain	Actn
0.37	0.08	0.87	−1.26	0.07	37
0.12	0.19	0.87	0.07	1.55	37
0.12	0.19	0.18	1.11	0.81	13
1.08	−1.44	−0.49	−1.54	−2.83	25
1.08	−1.44	−0.49	−1.26	1.37	25
1.28	−0.55	−0.13	0.07	−0.51	12
0.12	−0.57	0.18	1.11	0.59	15
−0.39	−0.24	0.87	−1.26	1.37	25
0.71	−0.54	−0.56	0.17	−0.96	34

Next, we solved independently instances in set A and set B using the general hyper-heuristic. Table 3 summarizes results for set A. Results are compared against those generated by the best single heuristics. Heuristics are grouped in subsets of four where the selection heuristic is common among them. Figures in cells indicate the percentage of problems solved with a particular number of extra objects (left column) when compared with the results provided by the best heuristic. For example, on the column labeled 'FFD', heuristics 5 to 8 solved 67.96% of the instances in set A with the same number of objects as the best heuristic for each given instance. It is interesting to see, if we focus in the results obtained by the hyper-heuristic, that 80.37% of problems were solved with no extra objects (percentage higher than any other set of heuristics). In 1.48% of problems, the hyper-heuristic obtained solutions with one object less than the best heuristic. It is important to emphasize that the best single heuristic is not always the same for all instances, but there is a tendency to achieve reasonable performance from those heuristics known to be good, in general, for problems of this kind. 'DJD' is in this case, and it comes second to the hyper-heuristic. However, other sets of heuristics show questionable performance since some of them need more than five extra objects to solve a high percentage of instances.

For set B, whose instances were not seen before by the hyper-heuristic, results are shown in Table 4. Again, the hyper-heuristic shows better performance than the best single heuristics with 76.92% problems with 0 extra objects and 4.81% of problems with one object less. The other groups of heuristics behave very much the same as their performance shown with set A.

4.2 Experiment Type II

In order to confirm the performance and roboustness of the model, other set of experiments were designed. Basically, the procedure is the same as Experiment Type I, but in this case, set B is used as the training set instead. The hyper-heuristic created is shown in Table 5. We label it HH-II. This hyper-heuristic comprises nineteen blocks. Another interesting observation is that the single heuristics composing the HH-II are different from those composing HH-I. Heuristic 13 (Filler+FFD+BL), however, is common in both, and appears in four different rules in the hyper-heuristic. Heuristic 28 (BF+BLLTR) appears also four times, and Heuristic 7 (FFD+BLLT) appears three times with rather different conditions.

Table 6 shows results when using hyper-heuristic HH-II for solving problems in set B. Percentage on solved problems within zero and one less extra bin is 83.69% (77.40%+6.29%). Only in 15.81% of problems, the hyper-heuristic uses one extra bin than the best single heuristic. Overall, results are again better than the subsets of heuristics.

Table 7 presents results for instances in set A when using hyper-heuristic HH-II. These instances had not seen before in the training phase. Results confirm the acceptable performance of the hyper-heuristic.

4.3 Experiment Type III

We generated additional 270 instances (group C) with the intention to compare performance of both hyper-heuristics generated in the two previous experiments. Table 8 presents the results. In addition, every single instance was solved using each of the combinations of selection and placement heuristics, keeping the best combination for each instance.

Table 3: **HH-I: Number of extra objects for problems in set A.**

		Heuristics									
		FF	**FFD**	**FFI**	**Filler**	**NF**	**NFD**	**BF**	**BFD**	**WF**	**DJD**
Obj.	**HH-I**	**1-4**	**5-8**	**9-12**	**13-16**	**17-20**	**21-24**	**25-28**	**29-32**	**33-36**	**37-40**
-1	**1.48**										
0	**80.37**	30.3	67.96	16.85	73.7	30.55	66.29	30.74	69.25	30.37	**74.81**
1	18.14	31.4	29.44	21.85	23.51	29.81	30.92	30.55	27.77	31.85	22.59
2		22.2	2.40	7.96	2.59	22.22	2.59	22.96	2.77	21.85	2.40
3		9.07	0.18	9.07	0.18	10.37	0.18	8.70	0.18	9.25	0.18
4		5.55		8.14		5.55		5.37		5.37	
5		1.11		9.81		1.29		1.48		1.11	
> 5		0.18		26.29		0.18		0.85		0.18	

Table 4: **HH-I: Number of extra object for problems in set B.**

		Heuristics									
		FF	**FFD**	**FFI**	**Filler**	**NF**	**NFD**	**BF**	**BFD**	**WF**	**DJD**
Obj.	**HH-I**	**1-4**	**5-8**	**9-12**	**13-16**	**17-20**	**21-24**	**25-28**	**29-32**	**33-36**	**37-40**
-1	**4.81**										
0	**76.92**	33	66.75	20.46	71.01	33.98	67.5	33.61	69.53	33.24	**71.66**
1	18.26	30.5	29.94	23.79	24.62	29.25	27.77	29.62	26.11	30.37	20.92
2		15.4	2.4	7.96	3.92	14.16	3.79	15.09	3.42	15.46	1.66
3		9.62	0.83	7.68	0.37	10.74	0.37	10.55	0.37	9.25	0
4		8.7	0.92	7.03	0.55	8.51	0.55	7.96	0.55	9.07	0.18
5		1.85	0.92	7.96		2.59		2.59		1.03	
> 5		0.74	1.20	25.09		0.74		0.55		0.55	

Table 5: **Experiment Type II: Hyper-heuristic (HH-II) produced by the GA.**

Huge	**Large**	**Medium**	**Small**	**Remain**	**Actn**
-0.95	−1.09	0.26	1.67	−0.02	8
0.34	0.14	0.1	1.92	−0.65	28
-0.28	−0.59	0.88	−0.38	1.32	13
−1.07	−0.41	0.45	−0.38	−0.72	15
−1.07	−0.41	−0.82	−0.35	−0.34	14
0.24	−0.38	0.88	−0.38	1.32	13
−0.28	−0.59	−1.6	−0.86	−0.5	2
−0.86	0.27	0.26	−0.38	1.32	13
−0.99	−1.61	−1.61	1.92	−0.65	28
0.63	−1.54	0.55	0.14	−0.82	7
−0.86	0.27	0.45	0.86	0.71	39
0.16	−0.14	−0.6	−2.14	−0.34	28
0.63	−1.54	0.55	0.15	−0.14	28
0.63	−1.54	0.55	0.14	−0.82	7
0.24	−0.38	−1.9	1.7	−0.25	7
−1.02	−1.15	0.4	−0.88	0.98	21
−0.04	1.71	1.35	−0.19	−0.76	13
0.14	−0.79	0.16	0.86	0.71	39
0.26	−0.5	−0.77	1.21	−0.28	3

It can be observed that behavior in both hyper-heuristic is very similar. Both solve around 3% of problems with one object less than the best heuristic, around 80% of problems with zero extra objects, and approximately 18% of problems with one extra object.

4.4 Analysis on the hyper-heuristics produced

Looking at the results, it is clear in all cases, that the method to form hyper-heuristics, and the hyper-heuristics themselves are efficient, at least with respect to the number of objects used for each instance. The GA-based procedure has found hyper-heuristics composed of a set of rules which associate the problem state to a combination of selection and placement heuristics. However, it is important to get a better feeling of the real advantages or the proposed approach, and the practical implications of using it. For example, regarding the computational cost for delivering solutions by our approach, it is slightly higher than the time used by the simple heuristics which run in just few seconds. Table 9 summarizes results on the performance of hyper-heuristics HH-I and HH-II with respect to the different testing sets we used (A, B, and C). Around 82% of problems in all sets are solved with zero or one less extra objects. In average, the best single heuristics solve around 75% of problems with zero extra objects, but the remaining 25% with one or more extra objects.

Results also confirm the idea behind hyper-heuristics that by exploiting the problem-specific features by means of choosing a set of heuristics which best adapt to that, a better performance can be achieved.

Table 6: HH-II: Number of extra objects for problems in set B.

		Heuristics									
		FF	**FFD**	**FFI**	**Filler**	**NF**	**NFD**	**BF**	**BFD**	**WF**	**DJD**
Obj.	**HH-II**	**1-4**	**5-8**	**9-12**	**13-16**	**17-20**	**21-24**	**25-28**	**29-32**	**33-36**	**37-40**
-1	**6.29**										
0	**77.40**	31.2	69.62	17.59	72.7	32.22	69.25	31.85	71.29	31.48	**74.81**
1	15.81	30.1	28.14	22.59	25	28.88	28.14	29.25	26.48	30	23.14
2		16.8	2.03	8.33	2.03	15.55	2.40	16.48	2.03	16.85	1.85
3		10.1	0	8.51	0	11.11	0	10.92	0	9.62	0
4		9.07	0.18	7.77	0.18	8.88	0.18	8.33	0.18	9.44	0.18
5		1.85		8.88		2.59		2.59		1.03	
> 5		0.74		26.29		0.74		0.55		0.55	

Table 7: HH-II: Number of extra objects for problems in set A.

		Heuristics									
		FF	**FFD**	**FFI**	**Filler**	**NF**	**NFD**	**BF**	**BFD**	**WF**	**DJD**
Obj.	**HH-II**	**1-4**	**5-8**	**9-12**	**13-16**	**17-20**	**21-24**	**25-28**	**29-32**	**33-36**	**37-40**
-1	**2.85**										
0	**77.77**	30	67.77	16.85	73.51	30.37	66.11	30.37	69.07	30	**74.62**
1	18.51	32.2	29.25	21.85	23.3	30.18	30.74	31.11	27.59	32.59	22.40
2		21.8	2.59	7.96	2.77	22.03	2.77	22.96	2.96	21.48	2.59
3		9.7	0.18	9.07	0.18	10.37	0.18	8.51	0.18	9.25	0.18
4		5.5		8.14		5.55		5.37		5.37	
5		1.11		9.81		1.29		1.48		1.11	
> 5		0.18		26.29		0.18		0.18		0.18	

Table 8: HH-I and HH-II: Number of extra objects for problems in set C.

			Heuristics									
			FF	**FFD**	**FFI**	**Filler**	**NF**	**NFD**	**BF**	**BFD**	**WF**	**DJD**
Obj.	**HH-I**	**HH-II**	**1-4**	**5-8**	**9-12**	**13-16**	**17-20**	**21-24**	**25-28**	**29-32**	**33-36**	**37-40**
-1	**2.59**	**2.96**										
0	**79.6**	**79.6**	68.1	75.37	46.66	74.44	67.40	74.07	66.48	73.88	67.59	77.59
1	17.7	17.4	31.4	24.25	40.92	25.19	32.22	25.55	33.51	25.74	32.03	22.03
2			0.37	0.37	11.66	0.37	0.37	0.37		0.37	0.37	0.37
3					0.74							

Table 9: Summary of Results: Hyper-heuristics, Best Single Heuristics and Testing Sets.

Problems	**Results**								
	HH-I			**HH-II**			**BSH**		
Extra Objects	**-1**	**0**	**1**	**-1**	**0**	**1**	**0**	**1**	> 1
Set A	1.48	80.37	18.14	2.85	77.77	18.51	74.81	22.4	2.58
Set B	4.81	76.92	18.26	6.29	77.4	15.81	71.66	20.92	1.84
Set C	2.59	79.6	17.7	2.96	79.6	17.4	77.59	22.03	0.37

5. CONCLUSIONS AND FUTURE WORK

This document has described experimental results in a model based on a variable-length GA which evolves combinations of condition-action rules representing problem states and associated selection and placement heuristics for solving 2D-Regular cutting stock problems. These combinations are called hyper-heuristics. Overall, the scheme identifies efficiently general hyper-heuristics after going through a learning procedure with training and testing phases. When applied to unseen examples, those hyper-heuristics solve most of the problems very efficiently, in fact, much better than the best single heuristic for each instance.

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 approach for other multidimensional problems.

6. ACKNOWLEDGMENTS

This research was supported in part by ITESM under the Research Chair CAT-010 and the CONACYT Project under grant 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 cuting 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] A. I. Hinxman. The trim-loss and assortment problems: A survey. *EJOR*, 5:8–18, 1980.

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

[14] 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.

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

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

[17] 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.

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

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

[20] P. Ross, J. M. Blázquez, S. Schulenburg, and E. Hart. Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics. *Proceedings of GECCO 2003*, pages 1295–1306, 2003.

[21] 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.

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

[23] 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.

[24] H. Terashima-Marín, A. Morán-Saavedra, and P. Ross. Forming hyper-heuristics with GAs when solving 2D-regular cutting stock problems. *Proceedings of the Congress on Evolutionary Computation*, pages 1104–1110, 2005.

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