Hyper-heuristics and Classifier Systems for Solving 2D-Regular Cutting Stock Problems

H. Terashima-Marín
ITESM-Center for Intelligent Systems
Av. E. Garza Sada 2501
Monterrey, NL, 64849 Mexico
terashima@itesm.mx

E. J. Flores-Álvarez
ITESM-Center for Intelligent Systems
Av. E. Garza Sada 2501
Monterrey, NL, 64849, Mexico
floresedgardo@yahoo.com

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

ABSTRACT

This paper presents a method for combining concepts of Hyper-heuristics and Learning Classifier Systems for solving 2D Cutting Stock Problems. 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. In this paper, the Hyper-heuristic is formed using a XCS-type Learning Classifier System which learns a solution procedure when solving individual problems. The XCS evolves a behavior model which determines the possible actions (selection and placement heuristics) for given states of the problem. When tested with a collection of different problems, the method finds very competitive results 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

General Terms

Algorithms

Keywords

Evolutionary Computation, Hyper-heuristics, Classifier Systems, 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 [10]. 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 [15]. For small combinatorial problems, exact methods like linear programming can be applied. However, when larger and more complex problems appear, exact solutions are not an option since the search space grows exponentially, and so 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 primary purpose of this paper is to explore an idea previously developed for one-dimensional bin packing [22], and extend that idea to solve 2D-rectangular cutting stock problems. There are similarities between the two problems, so heuristics developed for bin packing can also be used in the solution process for the 2D-cutting stock problem. The model uses a learning mechanism based on what is known in the Evolutionary Computation field, as Learning Classifier Systems (LCS). In their paper and also in our research, a particular type of LCS is used, the XCS, which is based on the accuracy of the prediction, and it is used to learn a set of rules which associates features in the problem state with various heuristics. This is the concept known as hyper-heuristic.

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. Given that the cutting problem has specific features, constraints and heuristics, this investigation introduces a method which combines the ideas explained above to tackle 2D-rectangular cutting stock problems. The method assembles 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 defined under the scheme of classifier systems. The classifiers are evolved using a Genetic Algorithm (GA), with the aim of proposing better heuristics, for solving the given cutting stock problem.

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 [17]. 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 [12]; the evolution of the problem with the objective of maximal production [13]; the applications and solutions to the CuSP problem [7]; 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 $s(a_i)\epsilon(0, X]$, from a set of cutting stock sheets (objects) of size X, 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 [7] 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 [15] 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 [24]: Genetic Algorithms [14], Evolutionary Strategies [21, 23], and Evolutionary Programming [1, 9]. From these types of Evolutionary Algorithms, Genetic Algorithms were developed by Holland and his associates in Michigan University between 1960 and 1970, and the first systematic and theoretical treatment is in Holland's Adaptation in Natural and Artificial Systems [14]. Later Goldberg [11] gave a summary of the applications done up to 1989. Recently, many sources related to the topic have been published [6, 8, 20]. Based on the same ideas, Holland also developed a learning model called Learning Classifier System (LCS) which is an evolutionary technique combined with reinforcement learning and other heuristics to produce adaptive systems. The technique has been applied to a wide variety of domains such as optimization, design, classification, control and many others [3]. A hyper-heuristic is formed by combining a set of simple heuristics, and in this research we use a special type of LCS, called XCS [25], to perform this task.

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 de 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. [22] and Hopper et al. [15].

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) [16].- 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) [18] .- 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 in order to consider rotation in the piece to place. These heuristics are called BLR and BLLTR.

3.2 Combining Heuristics with the XCS

This section first describes in detail the concepts of Learning Classifier Systems and Hyper-heuristics, and next our specific model combining both techniques.

3.2.1 LCS and Hyper-heuristics

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 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 solve 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.

3.2.2 Proposed Solution Model

The above two concepts were merged to solve 2D cutting stock problems, following previous work by Ross et al. [22] for one dimensional bin packing. An XCS type classifier system 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.

A rule determines the relationship between a condition and an action. The representation of a classifier is shown in Figure 2. This represents a description of a problem state. The height, width, area, and the percentage R of items to be cut in each category is computed.

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
  - **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
  - **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
  - **LA.-** Items from 1/3 up to 1/2 of object area
  - **HA.-** Items over 1/2 of object area
- **R.-** Ratio of items to be cut

The action segment contains the following information:

- **SCH.-** Selection Heuristic
- **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, shown also in Table 1. The percentages of items remaining to be cut are shown in Table 2.

The action was selected from all possible combinations of selection and placement heuristics, taking also into consideration the possibility of rotating an object by 90 degrees. Those combinations are shown in Table 3.

Table 1: Proportion of items.

Bits	Proportion
0 0	0-10%
0 1	10-20%
1 0	20-50%
1 1	50-100%

The general procedure of the method has the following steps:

- The XCS generates a random population of classifiers.
- The current problem state is matched against those rules.

Table 2: Percentage of items left to be cut.

Bits	% left to be cut
0 0 0	0-14%
0 0 1	14-28%
0 1 1	28-42%
0 1 0	42-56%
1 1 0	56-70%
1 0 0	70-84%
1 1 1	84-100%

Table 3: 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)

- 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.
- 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.
- Once an instance has been completely solved, the best classifiers are kept in the population and the solution process starts again for that instance.
- The procedure continues until a pre-established number of cycles is reached.

4. RESULTS AND DISCUSSION

This section presents the experiments carried out during the investigation and the results obtained. These results are compared against those obtained by the individual heuristics for two kinds of problem instances: the first benchmark

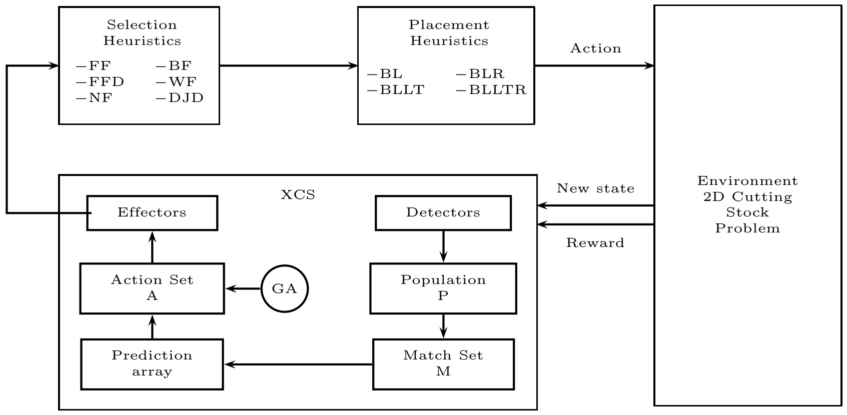

Figure 1: Model combining Hyper-Heuristics and the XCS Classifier System.

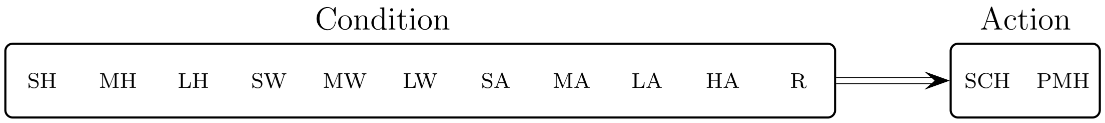

Figure 2: Structure of Classifiers.

set was taken from the literature (available from the OR-Library [2]), and the other one is composed by a set of randomly generated problems for which an optimal solution is known. The first set is an ensemble of 16 problems (guillotineable, labeled cgcut1 to cgcut3, and gcut1 to gcut13) plus 12 problems (non-guillotineable, labeled ngcut1 to ngcut12). They carry different features, for instance, problem cgcut1 is composed of very small pieces in comparison to the size of the objects. cgcut1 has 16 pieces, cgcut2 has 23 pieces, whereas cgcut3 has 62 pieces. These problems have been also used in other similar studies.

Comparison tables for all problems contain the label for each problem, the number of pieces n, the continuous lower bound L_0, the best solution proposed in the work by Martello and Vigo (MaVi) [19], the best result provided by the individual heuristic (BH), and the results by our approach SS and MS considering the single-step and multiple-step reward scheme in a classifier system, respectively. The continuous lower bound determines the minimum number of objects needed to satisfy the cutting of the demand of pieces, and is computed by the following formula:

$$L_o = \left\lceil \frac{\sum_{j=1}^{n} h_j w_j}{HW} \right\rceil \qquad (1)$$

where H represents the object height, W is the width object, and each one of the pieces $j \epsilon J = 1, ..., n$ has a height $h_j \leq H$ and width $w_j \leq W$.

All possible combinations of selection and placement heuristics were run for each instance of the testbed. In the results, BH is the best result obtained by one of those combinations. It is important to emphasize that sometimes the best result is provided by more than one combination, but we do not have space here to include results on the individual combinations for each problem instance.

Table 4 shows results for the first three problems. Problems cgcut1 and cgcut2 are rather easy, so that several of the the individual heuristics, and our approach, solve both problems with the optimal number of objects. For problem cgcut3 the lower bound is not reached, but there is an improvement with respect to the results reported by Martello and Vigo, since the number of objects is decreased from 23 to 20.

Table 4: Comparison with results for 2D instances from the literature.

Problem	cgcut1	cgcut2	cgcut3
n	16	23	62
L_0	2	2	16
MaVi	2	2	23
BH	2	2	20
HH-XCS-SS	2	2	20
HH-XCS-MS	2	2	20

Results on the next 13 problems (gcut1 to gcut13) are presented in Table 5. It can be observed that there is a decrease in the number of objects for problems gcut4, gcut6, gcut8, and gcut10 with respect to the best heuristic. For instance, in problem gcut6, the number of objects obtained with the best heuristic is 7, while our approach reports 6. Nevertheless, the number of objects is within one from the continuous lower bound. For problems gcut5, gcut9 and gcut13, which seem easier, the continuous lower bound is reached. For the rest of the problems, results are as good as those reported by the best heuristic, and for some of the instances, a slight improvement to the results reported by Martello and Vigo.

Table 6 shows results for the non-guillotineable problems. There is an improvement for problems ngcut2, ngcut3, ngcut10, ngcut11, and ngcut12. For the remaining problems, our approach performs as good as the best heuristic.

Table 5: Comparison with results for 2D instances from the literature.

Problem	gc1	gc2	gc3	gc4	gc5	gc6	gc7	gc8	gc9	gc10	gc11	gc12	gc13
n	10	20	30	50	10	20	30	50	10	20	30	50	32
L_0	3	5	7	12	3	5	9	12	3	6	7	13	2
MaVi	5	6	8	14	3	7	11	-	3	7	9	16	2
BH	4	6	8	14	3	7	11	14	3	8	8	16	2
HH-XCS-SS	4	6	8	**13**	3	**6**	11	**13**	3	**7**	8	16	2
HH-XCS-MS	4	6	8	**13**	3	**6**	11	**13**	3	**7**	8	16	2

Table 6: Comparison with results for 2D instances from the literature.

Problem	ngc1	ngc2	ngc3	ngc4	ngc5	ngc6	ngc7	ngc8	ngc9	ngc10	ngc11	ngc12
n	50	17	21	7	14	15	8	13	18	13	15	22
L_0	2	3	3	2	3	2	1	2	3	2	2	3
MaVi	3	4	3	2	3	3	1	2	3	3	2	3
BH	3	4	4	2	3	3	2	2	3	3	3	4
HH-XCS-SS	3	**3**	**3**	2	3	3	2	2	3	**2**	**2**	**3**
HH-XCS-MS	3	**3**	**3**	2	3	3	2	2	3	**2**	**2**	**3**

Additional randomly generated instances were produced to be fair in the comparison with the other methods. Three guillotineable problems were generated with 39, 79, and 120 pieces each, and continuous lower bound (which is also the optimal number of objects) of 5, 10, and 15 objects respectively. Other three non-guillotineable instances were produced with 33, 54, and 103 pieces each, and with the continuous lower bound of 5, 10, and 15 objects. Results for this collection of problems is presented in Table 7. For two of the non-guillotineable instances the optimal number of objects is found. Although, there is a decrease in the number of objects with respect to the best heuristic, the optimal number of objects was not reached for the rest of the problems. In general, the solution model presents a reasonable performance. The choice of using either the Single or Multi-Step Reinforcement Scheme makes no difference on the results.

Table 7: Comparison with results for 2D proposed instances.

Problem	G1	G2	G3	NG1	NG2	NG3
n	39	79	120	33	54	103
L_0	5	10	15	5	10	15
BH	6	13	19	6	11	16
HH-XCS-SS	6	**12**	**17**	**5**	**10**	16
HH-XCS-MS	6	**12**	**17**	**5**	**10**	16

4.1 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. This supports again the statement that non-direct representations for solving difficult optimization problems seems the right direction when using GAs. Direct encoding for very large problems requires long chromosomes. 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. However, we found variations in computational cost when comparing the single and multi-step rewarding mechanism. In general the single-step is faster than the multi step. For instance, for a selection of problems in the test sets, an important gap was observed. For cgcut3, single-step reports 9.4 seconds whereas multi-step takes 36 seconds. Problem cgcut12 is the case with the greatest difference since single-step takes around an hour (3613 seconds) and multi-step needs over 93 hours. For the other cases and even for the randomly generated instances, results were obtained in a matter of seconds.

By revising in detail hyper-heuristics which obtained the best result for each problem instance, it is worth pointing out that there are single heuristics that repeatedly appear in the solutions. But it is also interesting to observe that single heuristics with no very good performance when run individually, appear in good hyper-heuristics too. For instance, Figure 3 shows the percentage of usage for each single heuristic combination (labeled 1 to 40, and shown previously in Table 3) when solving problem cgcut3 with the single-step mechanism. Good individual combinations such as Filler+FFD plus BLLTR (16) and NFD plus BLR (22) are used very often. But all other heuristics also contribute in the solution (except combination 18). This confirms 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.

5. CONCLUSIONS AND FUTURE WORK

This document has described experimental results in a model which uses a classifier system to form hyper-heuristics, which is a combination of single heuristics. For the 2D-regular cutting stock problem, combinations of selection and placement heuristics were used. Overall, the scheme identifies efficiently combinations of heuristics that solve optimally many of the test instances used.

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.

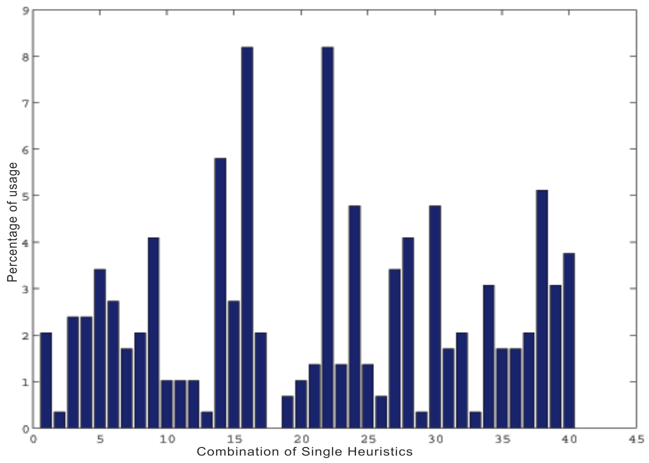

Figure 3: Selected heuristics by the XCS when solving problem cgcut3 with the Single-Step Rewarding Scheme.

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] L. Bull. *Applications of Learning Classifier Systems.* Springer, Berlin, 2004.

[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] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrtand Reinhold, New York, 1991.

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

[8] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Springer Verlag, Berlin, 2003.

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

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

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

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

[13] A. I. Hinxman. The trim-loss and assortment problems: A survey. *EJOR*, 5:8–18, 1980.

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

[15] 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.

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

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

[18] 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.

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

[20] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, Massachussets, 1996.

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

[22] 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.

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

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

[25] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, pages 149–175, 1995.