Building General Hyper-Heuristics for Multi-Objective Cutting Stock Problems

Juan Carlos Gómez[1] and Hugo Terashima-Marín[2]

[1]Department of Computer Science, KU Leuven, Belgium

[2]Center for Robotics and Intelligent Systems, Tecnológico de Monterrey, Campus Monterrey, Mexico

juancarlos.gomez@cs.kuleuven.be, terashima@itesm.mx

Abstract. In this article we build multi-objective hyper-heuristics (MOHHs) using the multi-objective evolutionary algorithm NSGA-II for solving irregular 2D cutting stock problems under a bi-objective minimization schema, having a trade-off between the number of sheets used to fit a finite number of pieces and the time required to perform the placement of these pieces. We solve this problem using a multi-objective variation of hyper-heuristics called MOHH, whose main idea consists of finding a set of simple heuristics which can be combined to find a general solution, where a single heuristic is applied depending on the current condition of the problem instead of applying a unique single heuristic during the whole placement process. MOHHs are built after going through a learning process using the NSGA-II, which evolves combinations of condition-action rules producing at the end a set of Pareto-optimal MOHHs. We test the approximated MOHHs on several sets of benchmark problems and present the results.

Keywords. Hyper-heuristics, multi-objective optimization, evolutionary computation, cutting problems.

Construyendo híper-heurísticas generales para problemas de corte multi-objetivo

Resumen. En este artículo se construyen Híper-Heurísticas Multi-Objetivo (MOHH por las siglas en Inglés), utilizando el algoritmo evolutivo multi-objetivo NSGA-II, para solucionar problemas de corte irregular en 2D empleando un esquema bi-objetivo; teniendo un balance entre el número de hojas usadas para ajustar un número finito de piezas y el tiempo requerido para realizar el acomodo de las piezas. Este problema es resuelto usando las MOHHs, cuya idea principal consiste en encontrar un conjunto de heurísticas simples que puedan ser combinadas para encontrar una solución general; donde una heurística simple es utilizada dependiendo de la condición actual del problema, en vez de aplicar una única heurística simple durante todo el proceso de acomodo. Las MOHHs son construidas a través de un proceso de aprendizaje evolutivo utilizando el NSGA-II, el cual evoluciona combinaciones de reglas condición-acción produciendo al final un conjunto de MOHHs Pareto-óptimas. Las MOHHs construidas son probadas en diferentes conjuntos de problemas y los resultados obtenidos son presentados aquí.

Palabras clave. Híper-heurísticas, optimización multi-objetivo, computación evolutiva, problemas de corte.

1 Introduction

Bin packing and cutting stock problems are well-known classical problems with many applications in areas like operational research, logistics and related subjects. The basic idea and the main goal are to fit a finite number of pieces into a minimum number of bins, subject to a practical set of restrictions. For small combinatorial problems, exact methods like linear programming can be applied. However, when larger and more complex problems appear, exact solutions are not feasible since the search space grows exponentially and so does the time to find the optimal solution. Various heuristic and approximate approaches that guarantee finding near optimal solutions have been proposed [5, 15]. However, no reliable method which can solve a large variation of instances of a given problem has been found.

Hyper-heuristic (HH) is a method used to define a high-level heuristic that controls low-level heuristics [2]. The hyper-heuristic decides when and where to apply each single low-level heuristic, depending on the state of a given problem and the search space. In recent work [17, 18], evolutionary approaches have been used to generate hyper-heuristics for the 2D regular and irregular cutting stock problems [19, 20, 21]. These methods assemble a combination of single heuristics, taking into account the quality of partial solutions provided by the single heuristics.

Until now, the majority of the works devoted to the problems of bin packing and cutting stock have been focused on mono-objective solutions like minimizing the trim loss or the number of bins used; however, these problems are naturally multi-objective, since several opposite objectives can be optimized at the same time. Just recently some works devoted to multi-objective cutting and packing problems have started to emerge [11, 16]. The present article intends to contribute with another perspective of solution for these problems. In particular, we focus on problems involving 2D cutting where two objectives need to be minimized: the number of sheets used to cut a finite number of pieces with irregular convex shapes and the time required to perform the placement of all the pieces inside the sheets.

The aim of this paper is to present a method based on the Multi-Objective Evolutionary Algorithm (MOEA) [6] NSGA-II [8] to approximate generalized Multi-Objective Hyper-Heuristics (MOHHs) in order to solve the cutting-stock problem described above. Our model is an adaptation of the one presented in [21]. Here we use NSGA-II with a variable-length representation, where this algorithm evolves combinations of condition-action rules through a learning process, producing at the end a set of Pareto-optimal MOHHs. Finally, we test the approximated MOHHs on several sets of benchmark problems. The results of the proposed model are truly encouraging.

The remainder of this paper is organized as follows. Section 2 describes with more detail the bi-objective cutting-stock problem. Section 3 gives a description of the NSGA-II algorithm, the hyper-heuristics and the single heuristics to be combined in the proposed solution model based on MOHH, which is presented in Section 4. This is followed by the experimental setup, the results and discussion in Section 5. Finally, Section 6 includes our conclusions and some ideas for future work.

2 Bi-objective Cutting Stock Problem

The cutting stock and packing problems are among the earliest problems in the literature of operational research. Since 1939, when L.V. Kantorovich [14] studied their applications in industry, an extensive literature on these problems' applications has developed. For example, B.L. Golden in [12] gives an abstract description of a variety of different solution methods; in [5] a number of solution methods are discussed; H. Dyckhoff [9] and Wäscher *et al*. [22] list a number of solution methods and applications, and present a systematic categorization of cutting and packing problems.

In this paper, our interest is to solve instances of problems considered as 2D cutting stock problems with convex irregular shape pieces (with up to 8 sides). We can formally define this type of problems in the following way: given a set $\mathbf{L}=(a_1,a_2,\ldots,a_n)$ of pieces to be cut, each one of size $s(a_i)\in(0,A_0]$, from a set of m cutting stock sheets of size A_0, the multi-objective goal of cutting the pieces from the sheets can be expressed as follows:

$$\text{minimize } z_1 = \sum_{i=1}^{m} y_i \tag{1}$$

$$\text{minimize } z_2 = \sum_{i=1}^{n} t_i \tag{2}$$

$$\text{s.t. } \sum_{i=1}^{n} s(a_i) \leq mA_0$$

where Expression 1 minimizes the number y_i of the sheets needed to fit all the pieces, and Expression 2 minimizes the time to place all the pieces inside those sheets, where t_i indicates the required time to put the piece a_i inside a sheet, using a given heuristic. The restriction avoids that the total area of the pieces is bigger than the area of the sheet. In this work we also consider that all the sheets must have the same size and the

same square shape. An example of the type of problems we are interested in can be seen in Figure 1.

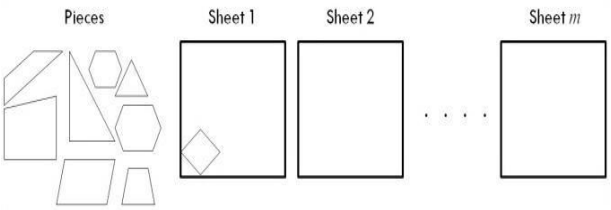

Fig. 1. Example of an irregular cutting problem

Intuitively, the two objective functions presented are of conflicting nature. While a large number of sheets allow a very fast placement of all the pieces, with $z_2 \rightarrow 0$, the trim of material will be also large; on the other hand, a solution with $z_1 \rightarrow 0$ will tend to require more time to place all the pieces in the best position without wasting material. This means that not a single solution x exists in the set of feasible solutions $\mathbf{X}$ that equally minimizes both functions z_1 and z_2. This is a vector optimization problem in which a solution $x \in \mathbf{X}$ is evaluated with respect to a vector $Z(x)=(z_1(x),z_2(x))$. Then, the solution of the problem consists in identifying all outcomes of the Pareto-set $\mathbf{S}$, defined as follows [6]:

Definition 1. ***Dominance:*** *$Z(x)$ is said to dominate $Z(x')$ iff $z_k(x) \leq z_k(x') \forall k=1,\ldots,K \wedge \exists k | z_k(x) \leq z_k(x')$. We denote the dominance of $Z(x)$ over $Z(x')$ with $Z(x) \langle Z(x')$.*

Definition 2. ***Efficiency, Pareto-optimality:*** *The vector $Z(x)$, $x \in \mathbf{X}$, is said to be efficient iff $\neg \exists Z(x'), x' \in \mathbf{X} | Z(x') \langle Z(x)$. The corresponding alternative x is called Pareto-optimal and the set of all alternatives is the Pareto-set $\mathbf{S}$.*

3 Methods

This section describes the Non-dominated Sorting Genetic Algorithm-II (NSGA-II), developed by K. Deb [8], and its adaptations to tackle the problem we have at hand. Here we also describe the concept of hyper-heuristics and the set of selection and placement heuristics used during the solution process with our model.

3.1 NSGA-II

In this paper we use the NSGA-II algorithm to build general MOHHs (this process is described below). The NSGA-II is part of the so-called Multi-Objective Evolutionary Algorithms (MOEAs) [6], which are methods suited well to solve multi-objective problems given their natural way of producing a set of Pareto-optimal non-dominated solutions within a single run. The NSGA-II is one of the best known MOEAs which is characterized by good behavior and used as a reference in many works on multi-objective optimization; so this was our motivation to integrate it in our model. At the initial step of this algorithm, a random parent population $\mathbf{P}_0$ of size N is created. The population is sorted based on non-domination. Each solution is assigned a fitness (or rank) equal to its non-domination level (1 is the best level, 2 is the next-best level, etc.). Thus, minimization of this rank is assumed. The normal binary tournament selection, recombination, and mutation operators are used to create an offspring population $\mathbf{Q}_0$ of size N. After that, a combined population $\mathbf{R}_t=\mathbf{P}_t+\mathbf{Q}_t$ is formed. Next, the population $\mathbf{R}_t$ is sorted according to non-domination, forming the fronts $\mathbf{F}=\mathbf{F}_1,\mathbf{F}_2,\ldots$. Since all previous and current population members are included in $\mathbf{R}_t$, elitism is ensured. Now, the solutions belonging to the best non-dominated set $\mathbf{F}_1$ are of the best solutions in the combined population and must be emphasized more than any other solution inside it. If the size of $\mathbf{F}_1$ is smaller than N, all the members of the set $\mathbf{F}_1$ are chosen for the new population $\mathbf{P}_{t+1}$. The remaining members of the population $\mathbf{P}_{t+1}$ are chosen from subsequent non-dominated fronts in the order of their ranking. Thus, solutions from the set $\mathbf{F}_2$ are chosen next, followed by solutions from the set $\mathbf{F}_3$, etc. This procedure is continued until no more sets can be accommodated. The new population $\mathbf{P}_{t+1}$ is now used for selection, crossover, and mutation to create a new population $\mathbf{Q}_{t+1}$, and the process is repeated until a stop criteria is satisfied. The NSGA-II main loop is shown in Algorithm 1. Inside this main process, we have two important sub-processes, the `fast-non-dom-sort` procedure (shown in Algorithm 2), which accomplishes a fast sort of the current

population in different non-domination levels; and the `crow-dist-assignment` procedure (shown in Algorithm 3), which measures the crowding-distance, a measure of density of solutions in the neighborhood used to preserve diversity among non-dominated solutions during the tournament selection and the population reduction phase.

Algorithm 1. NSGA-II main loop

```
1:  Q_{t+1} ← make-new-population(P_t)
2:  R_t ← P_t ∪ Q_t
3:  F ← fast-non-dom-sort(R_t)
4:  P_{t+1} ← ∅
5:  i ← 1
6:  while |P_{t+1}|+|F_i| ≤ N and F_i ≠ ∅ do
7:      crow-dist-assignment(F_i)
8:      P_{t+1} ← P_{t+1} ∪ F_i
9:      i ← i+1
10:     end while
11: sort(F_i, ≺_n)
12: P_{t+1} ← P_{t+1} ∪ F_i[1:(N-|P_{t+1}|)]
13: t ← t+1
```

Algorithm 2. `crow-dist-assignment`(**I**)

```
1:  l ← |I|
2:  for all i do
3:      I[i]_distance ← 0
4:      end for
5:  for all objective_m do
6:      I ← sort(I,m)
7:      I[1]_distance ← I[l]_distance ← ∞
8:      for i=2 to (l-1) do
9:          I[i]_distance ← I[i]_distance + (I[i+1]_m − I[i−1]_m) / (f_m^max − f_m^min)
10:     end for
11:     end for
```

Algorithm 3. `fast-non-dom-sort`(**P**)

```
1:  for all p∈P do
2:      S_p ← ∅
3:      n_p ← 0
4:      for all q∈P do
5:          if p ≺ q then
6:              S_p ← S_p ∪ q
7:          else if q ≺ p then
8:              n_p ← n_p+1
9:          end if
10:     end for
11:     if n_p=0 then
12:         p_rank ← 1
13:         F_1 ← F_1 ∪ p
14:     end if
15: end for
16: i ← 1
17:   while F_i ≠ ∅ do
18:       Q ← ∅
19:       for all p∈ F_i do
20:           for all q∈S_p do
21:               n_q ← n_q-1
22:               if n_q=0 then
23:                   q_rank ← i+1
24:                   Q ← Q ∪ q
25:               end if
26:           end for
27:       end for
28:       i ← i+1
29:       F_i ← Q
30:   end while
```

3.2 Hyper-Heuristics

Hyper-heuristics (HH) deal with the process of choosing a good single heuristic for solving the problem at hand. The idea is to discover a combination of single heuristics that can perform well on a whole range of problems and in such a way that one heuristic's strengths make up for the drawbacks of another [2, 4, 18]. The rationale is that there is no unique best single heuristic to solve well a wide range of instances of a given problem type, since certain problems may contain features that would enable a specific heuristic to work well but those features may not be present in other problems. Then, a combination of heuristics, selectively applied based on the features present in a problem, may work well on a large number of problems. HH have been used recently to solve a variety of problems like constraint satisfaction problems [1], timetabling [3] and scheduling [7]. The idea of hyper-heuristics is shown in Figure 2, where, starting from a problem E, in a given state space, we apply the best single heuristic for the current features of the problem, this will transform the problem into a new state E', with different features, where we can apply a new heuristic better appropriated for the new problem, this again will transform the problem into another state E'', with other features. The process is then repeated until a complete solution has been constructed.

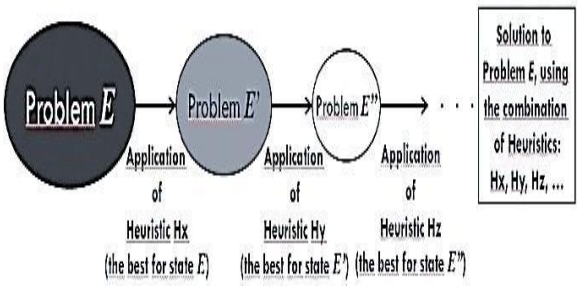

Fig. 2. Solution process for a problem using a combination of single heuristics

3.2.1 Single Heuristics

The single heuristics used to be combined in our model with MOHHs to solve the 2D convex irregular cutting-stock problem must define the exact location of the pieces inside the sheet. In this work two kinds of single heuristics were considered: one kind for selecting the pieces and sheets, and the other for placing the pieces into the sheets. The selection heuristics are shown below. Some of these heuristics are described in more detail in [13,18].

- **First Fit (FF):** Considers the opened sheets in turn in a fixed order and places the piece in the first one it fits.
- **First Fit Decreasing (FFD):** Sorts pieces in a decreasing order, and the largest one is placed according to FF.
- **First Fit Increasing (FFI):** Sorts pieces in an increasing order, and the smallest one is placed according to FF.
- **Filler + FFD:** Places as many pieces as possible within the open sheets. If at least one piece has been placed, the algorithm stops. Otherwise, the FFD algorithm is applied.
- **Next Fit (NF):** Uses the current sheet to place the next piece, otherwise opens a new one and places the piece there.
- **Next Fit Decreasing (NFD):** Sorts the pieces in a decreasing order, and the largest one is placed according to NF.
- **Best Fit (BF):** Places the piece in the opened sheet where it best fits (i.e., with the minimum waste).
- **Best Fit Decreasing (BFD):** Same as the previous one, but sorts the pieces in a decreasing order.
- **Worst Fit (WF):** Places the piece in the opened sheet where it worst fits (i.e., with the largest waste).
- **Djang and Fitch (DJD):** Places pieces in a sheet taking pieces by decreasing size until the sheet is at least one third full. Then, it initializes w, a variable indicating the allowed waste, and looks for combinations of 1,2,…,5 pieces producing a waste w. If any combination fails, it increases w accordingly.

The placement heuristics, described in detail in [21], are the following:

- **Bottom-Left (BLI):** The piece starts at the top right corner of the sheet, then slides down and left with a sequence of movements until no other movement is possible. The heuristic does not allow the piece to skip around another placed piece. It is a simple and fast heuristic.
- **Constructive Approach (CA):** The heuristic starts by placing the first piece at the bottom and left of the sheet. Then, the next piece is placed in one of the five positions: $(\dot{x},0),(0,\dot{y}),(\underline{x},\dot{y}),(\dot{x},\dot{y})$ and $(\dot{x},\underline{y})$, where $\dot{x}$, $\underline{x}$, $\dot{y}$ and $\underline{y}$ are the maximum and minimum coordinates of x and y in relation to the first piece. For each position, the next piece slides down and left, and the one that places the piece deepest (bottom and left) is chosen, except in special cases, such as when a hole is formed.
- **Constructive-Approach (Minimum Area) (CAA):** In this modification of the previous heuristic, the best position from the list is selected based on which one yields the bounding rectangle with minimum area, containing all pieces, and that fits in the bottom left corner of the object.
- **Constructive-Approach (Maximum Adjacency) (CAD):** With the first piece only the four corners of the sheet are considered. For the subsequent pieces, the possible points are the same as in CA. For each position in the list, the piece starts in that position, and its adjacency (i.e., the common boundary between its perimeter and the placed pieces and the sheet edges) is

computed. Then, the piece is slid down and left, and the adjacency is computed again. The position with the largest adjacency is selected as the position of the new piece.

Using the previous selection and placement single heuristics, there are 40 different combinations for the action to be taken in each step of the solution process; these actions are shown in Table 1.

Table 1. List of possible actions

	Selection	Placement	No.	Selection	Placement
1	FF	BLI	21	NFD	BLI
2		CA	22		CA
3		CAA	23		CAA
4		CAD	24		CAD
5	FFD	BLI	25	BF	BLI
6		CA	26		CA
7		CAA	27		CAA
8		CAD	28		CAD
9	FFI	BLI	29	BFD	BLI
10		CA	30		CA
11		CAA	31		CAA
12		CAD	32		CAD
13	Filler+	BLI	33	WF	BLI
14	FFD	CA	34		CA
15		CAA	35		CAA
16		CAD	36		CAD
17	NF	BLI	37	DJD	BLI
18		CA	38		CA
19		CAA	39		CAA
20		CAD	40		CAD

4 Building Multi-Objective Hyper-Heuristics

In this work, we adapted the evolutionary model proposed in [21]. The original model produces general hyper-heuristics using a simplified representation of the state of a problem. In this model, a chromosome consists of a number of points in the simplified state space, with each point being labeled with a given heuristic. Under this schema, a chromosome represents a complete recipe for solving a problem, using the simple algorithm: until the problem is solved, (a) determine the current problem state E, (b) find the nearest point to it, (c) apply the heuristic attached to the point, and (d) update the state. We use such ideas and apply them with the NSGA-II algorithm. The NSGA-II's task is to find a set **S** of Pareto-optimal chromosomes that are capable of obtaining solutions for a wider variety of problems, taking into consideration the tradeoff between the two minimization objectives defined in equations 1 and 2; the set of chromosomes in **S** are the MOHHs we are seeking.

4.1 Representation

Each chromosome is composed by a variable number of blocks. The initial number of blocks is randomly created and after that, the evolutionary process can create or delete blocks. In our model each block includes nine numbers. The first eight lie in the range 0,1 and represent the problem state; the label is the ninth number, which identifies a particular action (single heuristic). Then, a block can be seen as a labeled point. The NSGA-II's task is to create and evolve a certain number of such labeled points using the problem-solving algorithm defined above, where Euclidean distance is used to determine the nearest point.

In the problem state, the first three numbers are related to rectangularity, a quantity that represents the proportion between the area of a piece and the area of a horizontal rectangle containing it. These numbers represent the fraction of remaining pieces with high rectangularity 0.9, 1, medium rectangularity 0.5, 0.9, and low rectangularity 0, 0.5.

The fourth to seventh numbers are related to the area of pieces, and are categorized as follows (A_0 is the sheet area, A_p is the piece area): huge ($A_0/2<A_p$), large ($A_0/3<A_p<A_0/2$), medium ($A_0/4<A_p<A_0/3$), and small ($A_p<A_0/4$). The eighth number represents the fraction of the total number of pieces that remain to be placed. The label is selected from the combinations of single heuristics showed in Table 1. Figure 3 gives a graphical representation of a chromosome using the simplified state space. This simplified feature

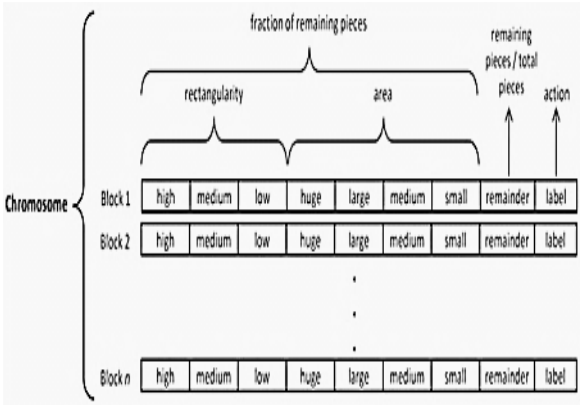

Fig. 3. Graphical representation of a chromosome using the simplified state space

space intends to represent the essential information about the geometry (rectangularity), and the occupied and free space of the problem's whole configuration in a given step of the solution process.

More features can be added or considered but with the corresponding increase in complexity. The current problem's state is computed in each step of the solution process, taking the information stored in the model about the pieces assigned, pieces free, and open (available) sheets.

4.2 Fitness Function

Each MOHH is evaluated using two fitness functions, one for the waste of space in the sheet and another for the total time needed to place the pieces on the sheet. The waste of space in a given sheet and the corresponding fitness function are defined as

$$W = 1 - \frac{\sum_{i=1}^{k} s(a_i)}{A_0} \tag{3}$$

$$FF_1 = \frac{\sum_{i=1}^{m} W_i^2}{m} \tag{4}$$

where k is the number of pieces inside the sheet, $s(a_i)$ the size of each piece, A_0 the size of the sheet, and m is the number of sheets used. The second fitness function is defined as

$$FF_2 = \sum_{i=1}^{n} t\left(h(a_i)\right) \tag{5}$$

where $t(h(a_i))$ is the required time to place the piece a_i using the heuristic h. During the NSGA-II process, when a new individual is created (in the first parent population or in the subsequent children populations), a set of 5 problems, randomly selected from the training set, is assigned to it. Then, the individual is evaluated for each of these problems using the previous fitness functions; the fitness values are added and averaged to obtain two final fitness values:

$$FFT_j(MOHH) = \frac{\sum_{i=1}^{5} FF_j(p_i)}{5}; j = 1,2 \tag{6}$$

where p_i is the i-th problem assigned to the individual, and the index j indicates the number of fitness function. During the evolution, if an individual from the parent population survives for the next generation, a new problem is assigned to it and the fitness functions are recomputed:

$$FFT_j^l(MOHH) = \frac{\left(FFT_j^{l-1} n_p\right) + FF_j(p)}{n_p + 1}; j = 1,2 \tag{7}$$

where FFTlj is the value of the j-th total fitness function for the generation l, np is the number of problems the individual has solved until now, and FFj(p) is the value of the j-th fitness function for the new problem. This task of assigning new problems to old individuals and recomputing their fitness values stays in effect during the whole evolutionary process.

4.3 Genetic Operators

In this work we use two uniform versions of cross-over and mutation: the first version works at the block level and the second one works with the internal elements of each block. In the block level cross-over, a random number of complete blocks is exchanged between two parent individuals to create two children individuals; since the number of blocks in each chromosome (individual) is

variable, the random number is less or equal to the shortest chromosome. In the block level mutation, given certain probability, a randomly selected block can be deleted from the chromosome or a randomly created block can be added to the chromosome. In the internal level cross-over and mutation, they select a random number of blocks, and for each block, a random number of values to be interchanged or mutate, depending on the operator. Examples of cross-over at the block level and the internal level are

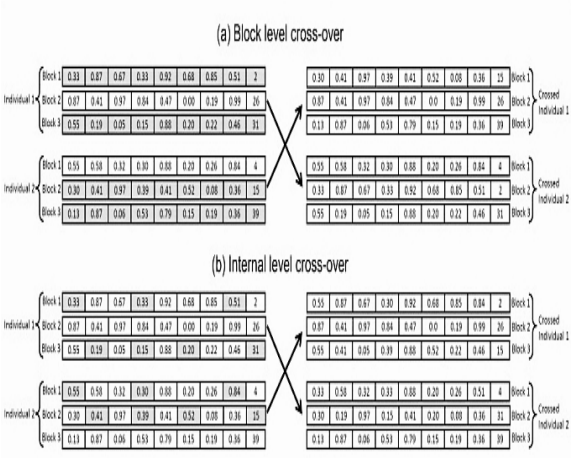

Fig. 4. Examples of cross-over operator at the block level (a) and the internal level (b)

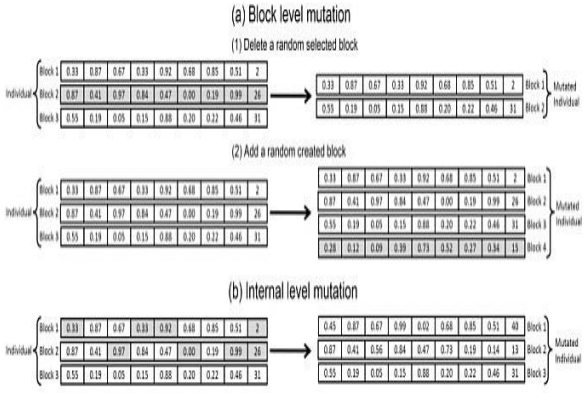

Fig. 5. Examples of mutation operator at the block level (a), by deleting a random selected block (1) and adding a random created block (2); and at the internal level (b)

shown in Figure 4. Examples of mutation at the block level and the internal level are shown in Figure 5.

5 Experimental Results

In this section we first present the problem instances used to perform the experiments with our model, then the experiments themselves, incl**u**ding a discussion of the results

5.1 Problem Instances

For this work we generated 18 different types of problems, with 30 instances for each type, totaling 540 instances. Their characteristics can be seen in Table 2. We also added a problem from the literature [10], which was scaled by a factor of 10 in order to have the sheet size of 300x300. Instances of type G have unknown optimal solutions, since they were produced from random **alterations of problems with known optimum.**

Table 2. Description of problem instances

Type	Sheets (size)	Pieces	Number of Instances	Optimum
Fu	300x300	12	1	unknown
A	1000x1000	30	30	3
B	1000x1000	30	30	10
C	1000x1000	36	30	6
D	1000x1000	60	30	3
E	1000x1000	60	30	3
F	1000x1000	30	30	2
G	1000x1000	36	30	unknown
H	1000x1000	36	30	12
I	1000x1000	60	30	3
J	1000x1000	60	30	4
K	1000x1000	54	30	6
L	1000x1000	30	30	3
M	1000x1000	40	30	5
N	1000x1000	60	30	2
O	1000x1000	28	30	7
P	1000x1000	56	30	8
Q	1000x1000	60	30	15
R	1000x1000	54	30	9

5.2 Experiments with the Proposed Model

In the beginning of each experiment, the problem instances are divided into the training set and the test set. The NSGA-II is performed with the training set only, until a termination criterion is met and the set **S** of general MOHH's has been evolved. All instances in the test set are then solved with each member of the set **S** and the results are recorded. In order to test the overall performance of the model, various experiments were designed:

- **Experiment I:** Instances are divided into two groups: Group 1 and Group 2. Group 1 is the training set formed by instance types A, B, C, D, E, F, G, H and I plus the Fu instance. After running our model over the training set, a set of Pareto-optimal MOHHs were obtained and each one was tested with Set 2 (instance types J, K, L, M, N, O, P, Q and R).
- **Experiment II:** This experiment is similar to Experiment I, except that the training and test sets are interchanged.
- **Experiment III:** This experiment takes the Fu instance and 15 instances from each problem type (from A through R) to form the training set. The remaining instances form the test set.
- **Experiment IV**: It is the same as Experiment III, except that the training and test sets are swapped.

We performed three independent runs of the evolutionary process with NSGA-II for each experiment. The settings used for the different runs were for run 1: 36 individuals and 80 generations; for run 2: 36 individuals and 240 generations; and for run 3: 108 individuals and 80 generations; with all the runs sharing a mutation probability of 0.1 (applied to each individual) and a cross-over probability of 0.9. A single run for one of the experiments, using the first settings, takes about 12 hours to approximate the set **S**, using our implementation in Java on a 2.8Ghz Core2Duro PC with 4Gb in RAM.

Each run produced as output a Pareto-optimal set $\mathbf{S}_{experiment;run}$ containing about 24 MOHHs for each experiment. Each one of the built MOHHs is used to solve all the instances in the test set, the results about the number of used sheets and the required time to place all the pieces are recorded and used later for comparisons. The sets produced for each run were then combined per experiment to obtain a unique single global set, having $\mathbf{S}_A=\mathbf{S}_{A1}\cup\mathbf{S}_{A2}\cup\mathbf{S}_{A3}$; $\mathbf{S}_B = \mathbf{S}_{B1}\cup\mathbf{S}_{B2}\cup\mathbf{S}_{B3}$; $\mathbf{S}_C=\mathbf{S}_{C1}\cup\mathbf{S}_{C2}\cup\mathbf{S}_{C3}$ and $\mathbf{S}_D=\mathbf{S}_{D1}\cup\mathbf{S}_{D2}\cup\mathbf{S}_{D3}$. Each global set obtained contained about 70 MOHHs; nevertheless, since the runs were performed independently, there existed a probability of having solutions inside the global set that were dominated by other solutions, according to the non-dominance criterion given in Definition 2. Then, we run a final non-dominance filtering and selected only the non-dominated MOHHs for each global set, leaving about 20 final solutions for each experiment.

Figure 6 presents the Pareto-optimal sets of MOHHs resulting from the combination of the three independent runs and the filtering using the non-dominance criterion. These graphs summarize the behavior of the selected MOHHs on solving all the instances in the test set. Given the definition of Equations 1 and 2, both objectives z_1 (number of sheets) and z_2 (time of placement) need to be minimized, then a point in the (minimum sheets, minimum time) corner would be the ideal solution; nevertheless, since the objectives are opposite to each other, that solution does not exist.

From these plots we can easily observe the existent trade-off between the number of used sheets and the time required to place the pieces. The points more to the left tend to spend more time in the placement process but using less sheets, and the ones more to the right tend to do the placement very fast but using more sheets. It is possible to observe how significant improvements in z_1 can be achieved for "similar" values of z_2, i.e., there are MOHHs that solve the problems using fewer sheets with a little increase of time.

Since it is complicated to integrate in a single table all the MOHHs built by the evolutionary process (about 20) for each experiment; we present in tables a selection of 12 MOHHs from each set **S**, corresponding to each experiment. We selected these MOHHs by taking the extremes of the fronts (the most right and the most left points in the above graphs) and the rest we selected randomly.

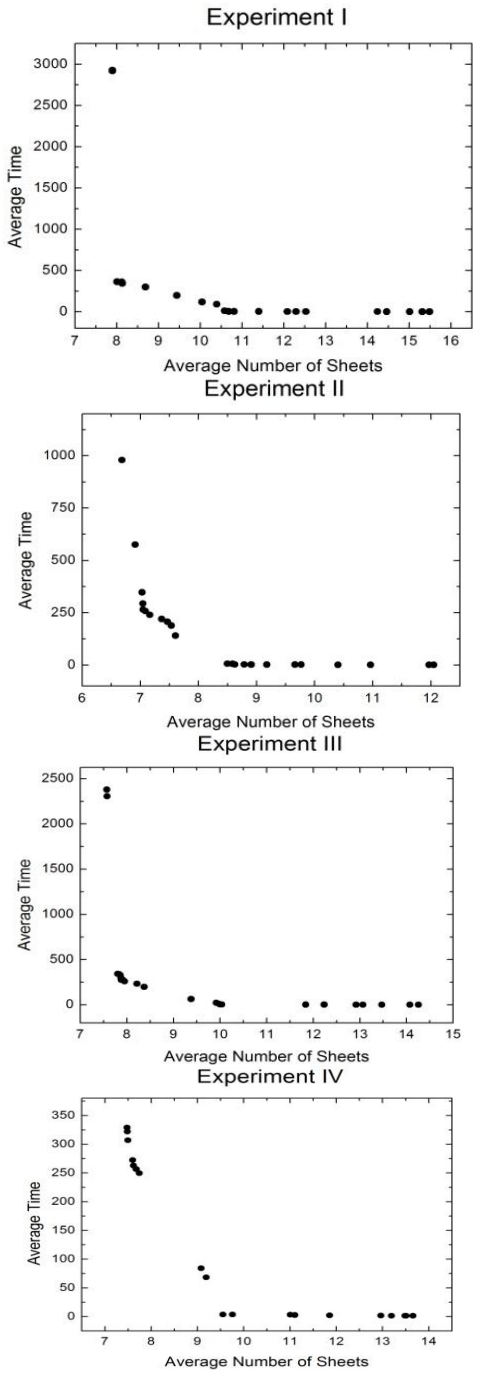

Fig. 6. Sets of non-dominated MOHHs (represented as points) forming the Pareto-fronts for Experiments I, II, III and IV

Tables 3, 4, 5 and 6 summarize the results for each one of the experiments, using the selected MOHHs from each set **S** as described above. The second row of the tables represents the average normalized time a MOHH needs to solve a problem, using its combination of single heuristics. The time is not in real time, but was measured by assigning a time score to each single heuristic, depending on the number of trials or movements it requires to place a piece inside a sheet; which in fact depends on the current configuration of the previous placed pieces in the sheet. These scores were assigned before the experiments by solving all the problems with every single heuristic to know the performance of each one for every problem, where the most expensive heuristic has a value of 100, and the rest of the heuristics have a fraction of this time. When applying the MOHH to solve a group of problems, for every heuristic used during the solution process its score is added and averaged at the end by the number of problems solved. The selected solutions are presented in the tables sorted in decreasing time.

Given a single problem instance, there is a single heuristic from the set of 40 heuristics that solves it using the least number of sheets. Taking this as a baseline, the rest of the rows in the tables show the number of extra sheets a MOHH needs to place all the pieces in comparison with that best single heuristic. The number in each cellis the percentage of problems where the MOHH needs from -1 to more than 3 sheets to place all pieces with respect to the collection of best single heuristics. It is possible to observe that when a small number of extra sheets (-1 or 0) is needed, the time to perform the placement is big, and the opposite: when the placement time is small, the number of extra sheets is big due to the conflicting objectives. This allows for a user to select the best solution in accordance to his/her particular needs: fastness (large number of extra sheets), precision (small number of extra sheets) or equilibrium.

For Experiment I, the built Pareto-front presents a gap (shown in top left plot of Figure 4 and Table 3) between the left most point and the rest of the points, this is due to the fact that the test set includes more diversity in the number of pieces and the optimum number of sheets, in comparison with the training set (as shown in Table 2). Then to solve a problem in the test set is

Table 3. Average placing time for the test set and percentage of problems where each selected MOHH needs from -1 to more than 3 sheets with respect to the collection of best single heuristics. Experiment I

HH_{A6}		HH_{A1}	HH_{A2}	HH_{A3}	HH_{A4}	HH_{A5}	HH_{A7}	HH_{A8}	HH_{A9}	HH_{A10}	HH_{A11}	HH_{A12}
91.68	**Time**	2926.06	363.26	299.75	197.89	118.11	11.58	4.37	3.56	2.58	2.17	1.63
%	**Sheets**	%	%	%	%	%	%	%	%	%	%	%
	-1	4.81			0.37							
0.37	0	87.03	83.7	72.96	41.11	9.25						
25.92	1	6.66	15.55	15.92	34.07	32.22	18.88	18.88	12.96	13.33	2.22	0.37
33.70	2	1.11	0.74		11.11	24.07	28.14	28.14	33.33	13.33	11.11	14.44
18.88	3				1.85	14.81	27.40	26.29	25.55	12.22	10.74	7.40
21.11	>3	0.37		11.11	11.48	19.62	25.55	26.66	28.14	61.11	75.92	77.77

Table 4. Average placing time for the test set and percentage of problems where each selected MOHH needs from -1 to more than 3 sheets with respect to the collection of best single heuristics. Experiment II

	HH_{B1}	HH_{B2}	HH_{B3}	HH_{B4}	HH_{B5}	HH_{B6}	HH_{B7}	HH_{B8}	HH_{B9}	HH_{B10}	HH_{B11}	HH_{B12}
Time	979.40	575.27	348.11	265.22	207.53	189.02	141.46	7.12	5.87	3.30	2.92	1.30
Sheets	%	%	%	%	%	%	%	%	%	%	%	%
0	97.41	80.07	70.47	69.37	52.39	57.19	28.04	10.70	8.85	7.38	0.36	0.73
1	2.58	13.65	19.55	20.29	26.56	16.23	45.38	26.93	28.78	25.83	31.36	11.43
2		4.79	8.48	8.48	8.11	10.70	20.66	28.78	29.15	27.30	22.87	12.91
3		1.47	1.47	1.84	7.74	8.11	5.90	19.55	19.92	20.66	16.23	12.17
>3					5.16	7.74		14.02	13.28	18.81	29.15	62.73

Table 5. Average placing time for the test set and percentage of problems where each selected MOHH needs from -1 to more than 3 sheets with respect to the collection of best single heuristics. Experiment III

	HH_{C1}	HH_{C2}	HH_{C3}	HH_{C4}	HH_{C5}	HH_{C6}	HH_{C7}	HH_{C8}	HH_{C9}	HH_{C10}	HH_{C11}	HH_{C12}
Time	2380.27	342.01	296.96	260.45	199.19	63.88	23.07	9.84	4.21	3.94	2.85	1.49
Sheets	%	%	%	%	%	%	%	%	%	%	%	%
-1	2.22				0.74							
0	91.48	79.25	72.59	71.85	46.29	8.14	5.92	4.81	4.44	4.81	0.37	0.37
1	6.29	14.81	21.48	18.51	35.55	38.88	25.55	24.07	24.07	22.96	18.88	6.29
2		5.18	4.81	6.66	6.66	30.37	25.18	26.29	26.29	25.92	15.92	14.07
3		0.74	1.11	2.22	8.51	14.44	24.44	25.18	24.44	25.55	12.96	8.88
>3				0.74	2.22	8.14	18.88	19.62	20.74	20.74	51.85	70.37

Table 6. Average placing time for the test set and percentage of problems where each selected MOHH needs from -1 to more than 3 sheets with respect to the collection of best single heuristics. Experiment IV

	HH_{D1}	HH_{D2}	HH_{D3}	HH_{D4}	HH_{D5}	HH_{D6}	HH_{D7}	HH_{D8}	HH_{D9}	HH_{D10}	HH_{D11}	HH_{D12}
Time	328.77	306.89	272.32	262.63	249.59	84.24	68.48	3.79	3.31	2.71	1.73	1.43
Sheets	%	%	%	%	%	%	%	%	%	%	%	%
0	75.27	74.16	69.37	68.63	65.68	8.48	11.43	4.05	0.73	0.73	0.36	0.36
1	20.29	20.66	21.03	21.40	20.66	25.83	21.03	24.35	18.81	13.65	5.53	5.53
2	3.69	4.79	7.74	7.38	7.01	35.05	32.47	30.62	20.66	15.86	14.02	13.28
3	0.73	0.36	1.47	2.21	4.42	20.66	19.92	20.29	19.55	17.71	11.80	10.70
>3			0.36	0.36	2.21	9.96	15.12	20.66	40.22	52.02	68.26	70.11

harder, and MOHHs solving these problems using few sheets would tend to increase the solving time faster. Then, in the case of Experiment II, we observe a more continuous Pareto-front (top right plot of Figure 4 and Table 4); because of the opposite reason, to solve a problem in the test set is easier given the less diversity, and MOHHs solving these problems using few sheets do not need to increase the solving time too much.

For Experiments III and IV, there is a combination of problems from all the types and the diversity is present in both the training and the test set. Given that, solutions from the test set are equally harder to solve than the ones from the training set. In that sense, Pareto-fronts for Experiments III and IV could be complementary, and if we remove the two most left points of the front III (bottom left plot of Figure 4 and Table 5), the rest of the MOHHs perform similarly in both Experiments III and IV (comparing bottom plots in Figure 4 and Table 5 and 6). The previous finding means that for Experiment IV, the algorithm is missing a couple of solutions more to the left, which in fact can be found if we add more runs of the algorithm.

Hyper-heuristics and multi-objective cutting are relatively novel areas, so the results presented in this work are in an early stage to be directly compared with other techniques, since there are still few works developed in the area, and no work on irregular cutting using the same objectives and the same set of problems.

6 Conclusions

In this paper we have described experimental results for a model based on the MOEA NSGA-II which evolves combinations of condition-action rules representing problem states and associated selection and placement heuristics for solving multi-objective 2D convex irregular cutting stock problems, where our goal was to minimize the (opposite) objectives of the number of sheets used to cut a set of pieces and the total time to place the pieces inside the sheets. These combinations of rules built by the model are called Multi-Objective Hyper-Heuristics (MOHHs). In general, the model efficiently builds the set of Pareto-optimal MOHHs after going through the training phase using NSGA-II and the set of training problems, and when applied to an unseen test set of problems, those built MOHHs solve the problems efficiently taking into account the trade-off between the two objectives. The results are truly encouraging, which could let the application of our model based on MOHH to be used in other areas and for other complex problems, where optimization of several opposite objectives is required. Several other considerations could be

taken into account when experimenting with MOHHs, like using different MOEAs in order to test and understand their behavior on producing the Pareto-front for different problems. In particular for the problem we tackled here, some interesting ideas for future work involve extending the proposed strategy to solve problems with more complex structure like 3D packing problems; including other objectives to be minimized, like the balance weight, the number of cuts or the heterogeneousness inside a sheet; or including other kinds of pieces with arbitrary shapes.

Acknowledgements

This research was supported in part by ITESM under the Research Chair CAT-144 and the CONACYT Project under grant 99695 and the CONACYT postdoctoral grant 290554/37720. A shorter version of the paper has already appeared in MICAI 2010.

References

1. **Bittle, S.A. & Fox, M.S. (2009).** Learning and using hyper-heuristics for variable and value ordering in constraint satisfaction problems. Annual Conference Companion on Genetic and Evolutionary Computation (GECCO'09), Montreal, Canada, 2209-2212.
2. **Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., & Schulenburg, S. (2003).** Hyper-heuristics: An Emerging Direction in Modern Research Technology. In Fred G. & Gary A. K. (Ed.), Handbook of Metaheuristics (457-474). Boston: Kluwer Academic Publishers.
3. **Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007).** A Graph Based Hyper-Heuristic for Educational Timetabling Problems. European Journal of Operational Research, 176(1), 177-192.
4. **Burke, E.K., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., & Vazquez-Rodriguez, J.A. (2009).** HyFlex: A Flexible Framework for the Design and Analysis of Hyper-heuristics. 4th Multi-disciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland, 790-797.
5. **Chen, C.H., Feiring, B.R., & Chang, T.C.E. (1994).** The Cutting Stock Problem, A Survey. International Journal of Production Economics, 36(3), 291-305.
6. **Coello, C.A., Van Veldhuizen, D.A., & Lamont, G.B. (2002).** Evolutionary Algorithms for Solving Multi- Objective Problems. New York: Kluwer Academic.
7. **Cowling, P.I., Kendall, G., & Soubeiga, E. (2000).** A Hyper-Heuristic Approach for Scheduling a Sales Summit. Selected papers from the Third International Conference on Practice And Theory of Automated Timetabling III, (PATAT'00), Konstanz, Germany, 176-190.
8. **Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2002).** A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. Parallel Problem Solving from Nature-PPSN VI, *Lecture Notes in Computer Science*, 1917, 849-858.
9. **Dyckhoff, H. (1990).** A Typology of Cutting and Packing Problems. European Journal of Operational Research, 44(2), 145-159.
10. **Fujita, K., Akagi, S., & Hirokawa, N. (1993).** Hybrid Approach for Optimal Nesting Using a Genetic Algorithm and a Local Minimisation Algorithm. 1993 ASME design technical conferences--19th Design Automation Conference, Albuquerque, USA, 477-484.
11. **Geiger, M.J. (2008).** Bin Packing Under Multiple Objectives - a Heuristic Approximation Approach. Fourth International Conference on Evolutionary Multi-Criterion Optimization, Matsushima, Japan, 53-56.
12. **Golden, B.L. (1976).** Approaches to the Cutting Stock Problem. AIIE Transactions, 8(2), 256-274.
13. **Hopper, E. & Turton, B.C.H. (2001).** An Empirical Study of Meta-Heuristics Applied to 2D Rectangular Bin Packing. Studia Informatica Universalis, 2(1), 77-106.
14. **Kantorovich, L.V. (1960).** Mathematical Methods of Organizing and Planning Production. Management Science, 6(4), 366-422.
15. **Lodi, A., Martello, S., & Monaci, M. (2002).** Two-dimensional Packing Problems: A Survey. European Journal of Operational Research, 141(2), 241-252.
16. **Muñoz, C., Sierra, M., Puente, J., Vela, C.R., & Varela, R. (2007).** Improving Cutting-Stock Plans with Multi-Objective Genetic Algorithms. Second International Work-conference on the Interplay between Natural and Artificial Computation, *Part I: Bio-inspired Modeling of Cognitive Tasks (IWINAC '07),* La Manga del Mar Menor, Spain, 528-537.

17. **Ross, P. (2005).** Hyper-Heuristics. In Burke, E. K. & Kendall, G.(Eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Methodologies (529-556). New York: Springer.
18. **Ross, P., Schulenburg, S., Marín-Blázquez, J.G., & Hart, E. (2002).** Hyper-Heuristics: Learning to Combine Simple Heuristics in Bin-Packing Problems. Conference on Genetic and Evolutionary Computation (GECCO '02), New York, USA, 942-948.
19. **Terashima-Marín, H., Flores-Álvarez, E.J., & Ross, P. (2005).** Hyper-Heuristics and Classifier Systems for Solving 2D-Regular Cutting Stock Problems. Conference on Genetic and Evolutionary Computation (GECCO '05), Washington, D.C., USA, 637-643.
20. **Terashima-Marín, H., Farías-Zárate, C.J., Ross, P., & Valenzuela-Rendón, M. (2006).** A GA-Based Method to Produce Generalized Hyper-Heuristics for the 2D-Regular Cutting Stock Problem. Conference on Genetic and Evolutionary Computation (GECCO '06), Seattle, USA, 591-598.
21. **Terashima-Marín, H., Ross, P., Farías-Zárate, C. J., López-Camacho, E., & Valenzuela-Rendón, M. (2010).** Generalized Hyper-Heuristics for Solving 2D Regular and Irregular Packing Problems. Annals of Operations Research, 179(1), 369-392.
22. **Wäscher, G., Haußner, H. & Schumann, H. (2007).** An Improved Typology of Cutting and Packing Problems. European Journal of Operational Research, 183(3), 1109-1130.

Juan Carlos Gómez received his Ph.D. degree from INAOE, México, in 2007. Currently he works as a Postdoctoral Fellow at the Department of Computer Science in the Katholieke Universiteit Leuven, Belgium. His research interests are Machine Learning, Information Retrieval, Evolutionary Computing and Data Mining.

Hugo Terashima-Marín received his Ph.D. degree from the ITESM, Campus Monterrey, Mexico, in 1998. Currently he is a Professor of Intelligent Systems and Computer Science and Director of Graduate Studies in ITESM, Campus Monterrey.

Article received on 09/02/2011; accepted on 03/11/2011.