Ann Oper Res (2010) 179: 369–392
DOI 10.1007/s10479-008-0475-2

Generalized hyper-heuristics for solving 2D Regular and Irregular Packing Problems

H. Terashima-Marín · P. Ross · C.J. Farías-Zárate · E. López-Camacho · M. Valenzuela-Rendón

Published online: 15 November 2008

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 a combination should outperform the single heuristics. This article presents a GA-based method that produces general hyper-heuristics that solve two-dimensional regular (rectangular) and irregular (convex polygonal) bin-packing problems. A hyper-heuristic is used to define a high-level heuristic that controls low-level heuristics. The hyper-heuristic should decide when and where to apply each single low-level heuristic, depending on the given problem state. In this investigation two kinds of heuristics were considered: for selecting the figures (pieces) and objects (bins), 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 of their type, considering their individual performance in various studies and also in an initial experimentation on a collection of benchmark problems. The GA included in the proposed model 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 for most of the cases. The testbed is composed of problems used in other similar studies in the literature. Some additional instances for the testbed were randomly generated.

H. Terashima-Marín (✉) · C.J. Farías-Zárate · E. López-Camacho · M. Valenzuela-Rendón
Tecnológico de Monterrey, Center for Intelligent Systems Monterrey, Nuevo León 64849, Mexico
e-mail: terashima@itesm.mx

C.J. Farías-Zárate
e-mail: a00789069@itesm.mx

E. López-Camacho
e-mail: eunice.lopez@itesm.mx

M. Valenzuela-Rendón
e-mail: valenzuela@itesm.mx

P. Ross
School of Computing, Napier University, Edinburgh, EH10 5DT, UK
e-mail: P.Ross@napier.ac.uk

Keywords Cutting Stock · Packing · Hyper-heuristics · Evolutionary computation

1 Introduction

Cutting stock and packing problems are widely studied because of their many applications ranging from clothing and metal to engineering and shipbuilding. The problems belong to the class of most difficult problems known as NP-hard (Garey and Johnson 1979). Given a set of figures (or pieces), the problem is to generate cutting patterns from sheets of stock material, or objects (or bins), that optimize certain objectives, such as to minimize the trim loss, or the number of objects used. In this particular investigation, problems involve packing 2D rectangular and 2D irregular (convex polygonal) pieces. Since precise requirements and constraints vary from industry to industry, many different approaches and techniques have been proposed for solving the problem (Hopper and Turton 2001b). 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 to find the optimal solution. Various heuristic and approximate approaches that guarantee finding near optimal solutions have been proposed. However, no reliable method that can solve all instances of a given problem has been found. In general, methods work well for some instances, but not for all.

The aim of this paper is to present a GA-based model that produce general hyper-heuristics for solving 2D bin-packing problems. The types of problems considered are the ones containing rectangular pieces, on one side, and irregular (convex polygonal) on the other. A hyper-heuristic is used to define a high-level heuristic that controls low-level heuristics (Burke et al. 2003a). 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 of the objective function, and so on. Selecting a particular heuristic is dynamic, and depends both on the problem state produced by the previous heuristic applied and on the search space to be explored at that point of time. In recent work which is based on the research by Ross et al. (2002) on uni-dimensional bin-packing, evolutionary approaches have been used to generate hyper-heuristics for the 2D Regular Cutting Stock Problems. Terashima-Marín et al. (2005a) use the XCS Classifier System to generate the hyper-heuristics, and Terashima-Marín et al. (2005b) use a GA with integer representation to solve the same problem. In other related work (Terashima-Marín et al. 2006), a model based on a Genetic Algorithm with variable-length representation to produce hyper-heuristics is described. All these approaches deliver very competitive results for a set of different problem instances, in fact beating results produced by the single heuristics. These methods assemble a combination of single heuristics (each concerned with selection and placement of a piece), and this combination takes into account the quality of partial solutions provided by the single heuristics. Based on the good results provided by the model reported by Terashima-Marín et al. (2006), this paper presents a more extensive investigation that confirms the good performance when applied to 2D regular instances, and also presents an extension that handles 2D irregular (convex polygonal) bin-packing problems with the same model. Results of the proposed model on both types of problems are truly encouraging.

The paper is organized as follows. Section 2 describes the general cutting and packing problems, and the particular features of the 2D bin-packing problem tackled in this investigation. Section 3 presents the solution method proposed and its justification. This is followed

by the experimental setup for both the regular and irregular approaches, the results, their analysis and discussion in Sect. 4. Finally, in Sect. 5 we include our conclusions and some ideas for future work.

2 Cutting Stock and Packing Problems

The Cutting Stock Problem (CuSP) is among the earliest problems in the literature of operational research. In 1939, Kantorovich studied obvious applications in all the industries whose products were in a flat sheet form; this research was published in Kantorovich (1960). An extensive literature on the CuSP and its applications has developed since. For example, Golden (1976) gives an abstract description of a variety of different solution methods; Dyckhoff (1990) lists a number of solution methods and applications; Cheng et al. (1994) discusses a number of solution methods.

Given a set $L = (a_1, a_2, \ldots, a_n)$ of pieces to be cut, each one of size $s(a_i) \epsilon (0, A_o]$, from a set of cutting stock sheets (objects) of size A_o, the problem is to find cutting patterns, in such a way that the solution minimizes the number of objects used. 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 (1990) proposed a systematic categorization of cutting and packing problems. His survey integrates a general system of 96 problems for the Cutting Stock with four main features and their subtypes as follows: Dimensionality: 1, 2, 3 or n; Assignation form (either all the larger objects and a selection of small figures (B), or a selection of large objects and all the small figures (V)); Assortment of large objects (one object (O), identical shapes (I), or different Shapes (D)); and Assortment of small figures (few figures of different shapes (F), many figures of different shapes (M), many figures of few of different and incongruent shapes (R), or congruent shapes (C)).

Our investigation in this paper is restricted to Cutting Stock Problems of two dimensions. 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 to be cut in the stock material (V). The stock material will have identical shapes (I); and the investigation will be done both for rectangular shapes (C) and for few figures of different shapes (F). Thus we will concentrate on 2VIC and 2VIF packing problems, in Dyckhoff's scheme. According to a new and more extensive typology recently proposed by Wäscher et al. (2007) the problems fall into the categories of two-dimensional, input (value) minimization, weakly heterogeneous assortment of small items, identical large objects, and regular (rectangular and polygonal) small items. The problems are considered as Single Bin-Size 2D Bin-Packing Problems.

3 Solution approach

Evolutionary Computation has been used in relatively few CuSP investigations. Hopper and Turton (2001a, 2001b) have presented an empirical study on the usage of Meta-Heuristics for solving 2D Rectangular Bin-Packing Problems. Other investigations on this topic and related to solving Packing Problems with GAs include (Kröger 1995; Reeves 1996; Dowsland et al. 2006). A recent investigation for creating hyper-heuristics with Learning Classifier Systems for solving packing problems has been published by Marín-Blázquez

and Schulenburg (2007). Evolutionary Computation usually includes several types of evolutionary algorithms (Wilson and Keil 1999): Genetic Algorithms (Holland 1975; Goldberg 1989), Evolutionary Strategies (Rechenberg 1973; Schwefel 1981), and Evolutionary Programming (Fogel et al. 1966; Banzhaf et al. 1998). In this paper we use a GA (Holland 1975; Goldberg 1989) with variable length chromosomes, basically a *messy*-GA (Goldberg et al. 1989).

3.1 The set of selection heuristics

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 (regular and irregular), 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: one kind for selecting the figures and objects, and the other for placing the figures into the objects. Some of the heuristics were taken from the literature, others were adapted, and some other variations were developed by us. We chose the most representative heuristics of their type, considering their individual performance as presented in related studies and also as shown by an initial experiment on a collection of benchmark problems. The selection heuristics used in this research are:

- First Fit (FF). Consider the opened objects in turn in a fixed order and place the item in the first one where it fits.
- First Fit Decreasing (FFD). Sort pieces in decreasing order, and the largest one is placed according to FF.
- First Fit Increasing (FFI). Sort pieces in increasing order, and the smallest one is placed according to FF.
- Filler + FFD. This places as many pieces as possible within the open objects. If at least one piece has been placed, the algorithm stops. The FFD algorithm is applied, otherwise.
- Next Fit (NF). Use the current object to place the next piece, otherwise open a new one and place the piece there.
- Next Fit Decreasing (NFD). Sort the pieces in decreasing order, and the largest one is placed according to NF.
- Best Fit (BF). This places the item in the opened object where it best fits, that is, in the object that leaves minimum waste.
- Best Fit Decreasing (BFD). Same as the previous one, but sorting the pieces in decreasing order.
- Worst Fit (WF). It places the item in the opened object where it worst fits (that is, with the largest available room).
- Djang and Fitch (DJD). It places items in an object, taking items by decreasing size, until the object is at least one-third full. Then, it initializes w, a variable 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 in more detail by Ross et al. (2002) and Hopper and Turton (2001b).

3.2 The set of placement heuristics

The heuristics used at this stage of the solution process differ slightly depending on the type of problem being solved. Although heuristics for placing regular pieces are of use when

placing irregular ones, they have to be modified accordingly. The following subsections present the heuristics used when solving each kind of problem.

Placement heuristics for 2D regular packing

The placement heuristics belong to the class of bottom-left heuristics, that is, they push pieces towards the bottom and left in the layout, using a sliding technique. The placement heuristics we used are:

- Bottom-Left (BL) (Jakobs 1996). The piece starts at the upper right corner of the object, then it slides vertically all the way down until it hits another piece; it then slides horizontally to the left in a straight line as far as possible. This sequence of down and left movements is repeated until the piece reaches a stable position.
- Improved-Bottom Left (BLLT) (Liu and Teng 1999). It is similar to the above heuristic, but if the downward movement hits a piece, then the subsequent leftward movement slides the piece along the upper contour of the already-placed pieces. Thus, unlike BL, the movement is not simply rectilinear.

From these, two new heuristics were generated to also consider rotations of the piece to place. These heuristics are called BLR and BLLTR.

Placement heuristics for 2D irregular packing

The heuristics used for this problem are based on the bottom-left heuristics explained above and the constructive approach presented by Hifi and MHallah (2003). They are the following:

- Bottom-Left (BLI). This is the best known heuristic of its type, and a modification to the bottom-left heuristic used for rectangular pieces. The piece starts at the top right corner of the object and it slides down and left with a sequence of movements until no other movement is possible (see Fig. 1). If the final position does not overlap the object boundaries, the piece is placed in that position. The heuristic does not allow a piece to skip around another placed piece. The good performance of this heuristic greatly depends on the initial ordering of the pieces as reported by Dowsland et al. (1998, 2002). Its advantage lies in its speed and simplicity.
- Constructive Approach (CA). This heuristic is based on the work presented by Hifi and MHallah (2003). The heuristic starts by placing the first piece at the bottom and left of the object. Then, the next piece is placed in one of the five positions: $(\bar{x}, 0)$, $(0, \bar{y})$, $(\underline{x}, \bar{y})$, $(\bar{x}, \bar{y})$ and $(\bar{x}, \underline{y})$, where $\bar{x}$, $\underline{x}$, $\bar{y}$, and $\underline{y}$ are the maximum and minimum coordinates in x and y in relation to the first piece (see Fig. 2). Given that some positions might coincide, each position appears only once in the list. For each position in the list, the next piece slides vertically and horizontally following down and left movements as shown in Fig. 3. Positions with overlapping pieces or exceeding the object dimensions are discarded. All others are considered as candidate positions, and the one that places the piece deepest (bottom and left) is chosen, except in special cases such as when a hole is formed (Hifi and MHallah 2002). In the implementation of the heuristic, we also added the four corners of the object as candidate positions. Using the corners as a departure point to slide the piece bottom and left, can make it possible to reach into certain gaps between pieces, gaps that would not reachable if only the five initial positions are considered. The approach uses simple geometric operators, avoiding more sophisticated computations such as the convex hull.

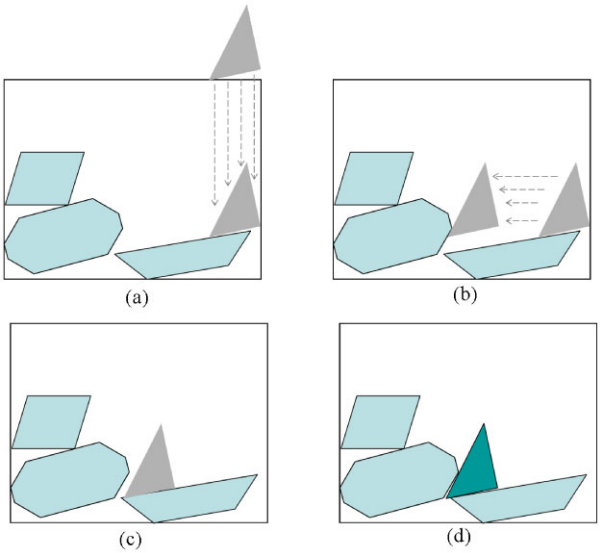

Fig. 1 Bottom-left heuristic for irregular pieces

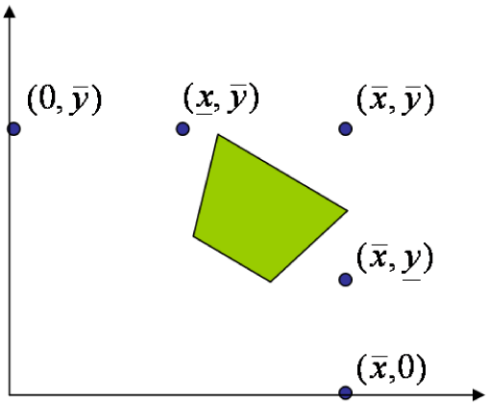

Fig. 2 Positions to be considered in the Constructive Approach

– Constructive-Approach (Minimum Area) (CAA). This is a modification of the previous heuristic. The variation consists of selecting the best position from the list 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. The rectangle is shown in Fig. 4 and its area is computed with the product of the maximum horizontal coordinate and the maximum vertical coordinate of all pieces already placed plus the new piece to be located in the proposed position. Figure 5 shows the rectangle with minimum area for two different proposed positions for a single piece. This criterion was chosen based on the idea of selecting a point with which all pieces are deepest (bottom and left), not only the last piece.
– Constructive-Approach (Maximum Adjacency) (CAD). The idea behind this heuristic is based on the approach suggested by Uday et al. (2001). In this investigation however, when the first piece is to be placed, only the four corners of the object are considered. For the subsequent pieces, the *possible points* are the same as those as in the constructive

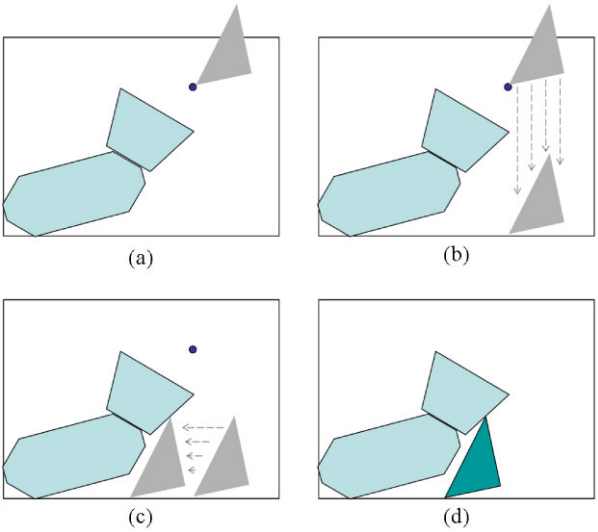

Fig. 3 Constructive Approach Heuristic

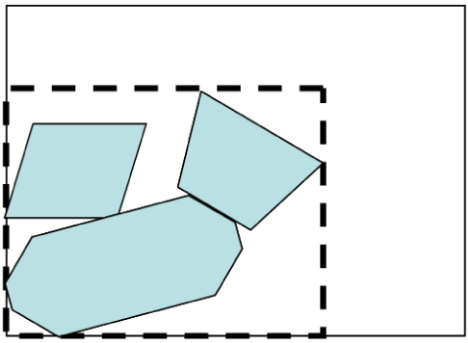

Fig. 4 Rectangle with minimum area, located at the bottom-left corner and containing all pieces so far

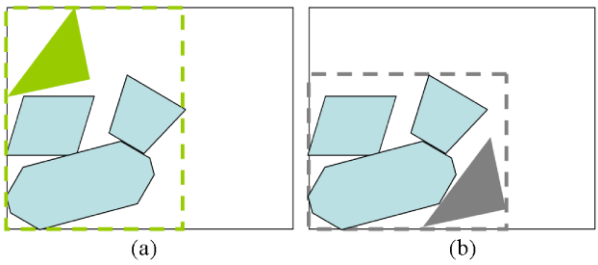

Fig. 5 Candidate rectangles when locating a piece

approach. Each position in the list is evaluated twice: first, the piece starts in that position and its adjacency (that is, the common boundary between its perimeter and the placed pieces and the object 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.

In order to implement the last three placement heuristics, it was necessary to consider several aspects related to computing the closest point where a piece can approach another; computing the adjacency between two pieces; deciding if two pieces intersect each other; computing if a piece is inside another; and deciding if two pieces are equal (given they are in different positions). For these heuristics, rotation was also important. For each position in the list of candidate positions, each heuristic tests different rotations for a given figure. We ran preliminary studies to determine a rotation scheme used in the investigation. The first one rotates each piece by multiples of 90 degrees, that is, 0, 90, 180 and 270. The second approach rotates each piece in multiples of 5 degrees. Interestingly, this second approach showed a very marginal improvement compared to the first approach, and had the disadvantage of the extra computational cost.

3.3 Combining heuristics with the proposed GA

Hyper-heuristics are motivated by the aim of providing a more general procedure for optimization (Burke et al. 2003a). Meta-heuristics methods usually solve problems by operating directly on the problem. Hyper-heuristics deal with the process of choosing good heuristics 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 and in such a way that one heuristic's strengths make up for the drawbacks of another. 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 an excessively 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 enable a specific heuristic to work well, but those features may not be present in other problems and might hinder other heuristics. But a combination of heuristics, selectively applied based on the features present in a problem, may work well on a wide range of problems.

The concept has received significant attention in recent years, and has been implemented to tackle various problems in different domains. The work by Bai et al. (2008) presents a combination of heuristic, meta-heuristic and hyper-heuristic approaches for inventory control and allocation in the fresh produce industry. Additional work related to hyper-heuristics, bin-packing methods, and their application to other optimization problems is reported by Dowsland et al. (2007), Burke et al. (2003b, 2006, 2007), Kendall et al. (2004), and Poli et al. (2007). The solution model proposed here uses features from work by Ross et al. (2003), for uni-dimensional bin-packing problems. In our research, a GA with variable-length individuals is used to find a combination of single heuristics (selection and placement) to solve efficiently a wide variety of instances of 2D Regular and Irregular packing problems. The basic idea is to have a simplified representation of the state of the problem at each step as a solution is constructed. A chromosome consists of a number of points in this simplified state space, each point being labelled with a selection heuristic and a placement heuristic. Given a problem state P in the simplified state space, find the closest point in the chromosome and apply the selection and placement heuristics recorded on the point's label. This will transform the problem to a new state P', and the process is repeated until a complete solution has been constructed.

A chromosome therefore represents a complete recipe for solving a problem, using this simple algorithm: until the problem is solved, (a) determine the current problem state P, (b) find the nearest point to it, (c) apply the heuristic attached to the point, and (d) update

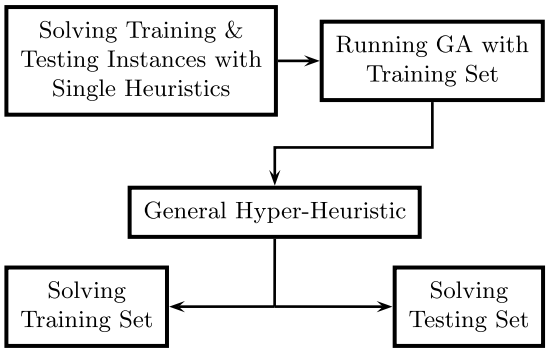

Fig. 6 Solution model

the state. The GA's task is to find a chromosome that is capable of obtaining good solutions for a wider variety of problems; the chromosome is the hyper-heuristic that we seek.

The available problem instances are divided into a training and a testing set. To evaluate the individual heuristics, each combination of a single selection heuristic and a single placement heuristic is applied to all problems, and the best is recorded for later comparison. Then the GA is used with the training set only, until a termination criterion is met and a general hyper-heuristic has been evolved. All instances in both the testing and training sets are then solved with this general hyper-heuristic and the results are recorded. The complete process is shown in Fig. 6.

Representation. Each chromosome is composed of a series of *blocks*. Each block j includes nine numbers. The first eight represent an instance of the problem state. For a given problem state, these eight numbers all lie in the range 0 to 1, so a block can be regarded as a labelled point in the eight-dimensional unit cube. The label is the ninth number, which identifies a particular pairing of a selection heuristic and a placement heuristic. The GA's task is to create an initially-unspecified number of such labelled points, and the problem-solving algorithm is simply this: given a point representing the current problem state, find the nearest labelled point, apply the heuristic attached to it, and repeat until all items in the problem have been selected and placed. To determine 'nearest' we use Euclidean distance. If we only allowed labelled points to lie inside the unit cube, there would be possible edge effects because the nearest labelled point to a corner would 'cover' less volume than interior points. To counteract this, we permit labelled points to lie anywhere in the $(-3, 3)^8$ cube, although the problem state can still only be in the unit cube. Experience suggests that the slight asymmetry of this, being centered on the origin rather than on the center of the unit cube, does not matter.

For the regular case, the initial four numbers in the eight-number state space representation indicate the fraction of pieces that remain to be packed according to the following categories (Ao is the object area, Ap is the item area): h_j, huge items ($Ao/2 < Ap$); l_j large items ($Ao/3 < Ap \leq Ao/2$); m_j, medium items ($Ao/4 < Ap \leq Ao/3$); and s_j, small items ($Ap \leq Ao/4$). The following three numbers indicate the fraction of items that remain to be packed according to a categorization based on the height (Ho is the object height, Hp is the item height): L_j large items ($Ho/2 < Hp$); M_j medium items ($Ho/3 < Hp \leq Ho/2$); and S_j, small items ($Hp \leq Ho/3$). The eighth number, r_j, represents the fraction of the total items that remain to be packed. The ninth number is an integer indicating the combination of heuristics (selection and placement), associated with this instance, see Table 1.

For the irregular case, the first three numbers of the eight are instead related to rectangularity, a quantity that represents the proportion between the area of a piece and the area

Table 1 Representation of actions (Regular Bin-packing)

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

of a horizontal rectangle containing it. The first number represents the fraction of remaining pieces with high rectangularity, in the range of 0.9 to 1 inclusive. The second corresponds to those of medium rectangularity, in the range 0.5 to 0.9. The third corresponds to low rectangularity, from 0 to 0.5. The fourth to seventh numbers are related to the area of pieces, and are categorized in a similar way as the representation used for regular problems explained above: huge, large, medium and small pieces. The eighth number represents fraction of the total number of pieces that remain to be packed.

The action was selected from all possible combinations of selection and placement heuristics, considering also the possibility of rotating the items. There are 40 combinations for each type of problem (regular or irregular) and they are shown in Tables 1 and 2.

The fitness function. The quality of a 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

$$P_u = \frac{\sum_{j=1}^{n} Ap_j}{Ao} \tag{1}$$

Table 2 Representation of actions (Irregular Bin-packing)

Actn	Selection	Placement	Actn	Selection	Placement
1	First Fit (FF)	BLI—Bottom Left (Irregular)	21	Next Fit Decreasing (NFD)	BLI—Bottom Left (Irregular)
2		CA—Constructive	22		CA—Constructive
3		CAA—Constructive-M. Area	23		CAA—Constructive-M. Area
4		CAD—Constructive-M. Adjacency	24		CAD—Constructive-M. Adjacency
5	First Fit Decreasing (FFD)	BLI—Bottom Left (Irregular)	25	Best Fit (BF)	BLI—Bottom Left (Irregular)
6		CA—Constructive	26		CA—Constructive
7		CAA—Constructive-M. Area	27		CAA—Constructive-M. Area
8		CAD—Constructive-M. Adjacency	28		CAD—Constructive-M. Adjacency
9	First Fit Increasing (FFI)	BLI—Bottom Left (Irregular)	29	Best Fit Decreasing (BFD)	BLI—Bottom Left (Irregular)
10		CA—Constructive	30		CA—Constructive
11		CAA—Constructive-M. Area	31		CAA—Constructive-M. Area
12		CAD—Constructive-M. Adjacency	32		CAD—Constructive-M. Adjacency
13	Filler + FFD	BLI—Bottom Left (Irregular)	33	Worst Fit (WF)	BLI—Bottom Left (Irregular)
14		CA—Constructive	34		CA—Constructive
15		CAA—Constructive-M. Area	35		CAA—Constructive-M. Area
16		CAD—Constructive-M. Adjacency	36		CAD—Constructive-M. Adjacency
17	Next Fit (NF)	BLI—Bottom Left (Irregular)	37	Djang and Finch	BLI—Bottom Left (Irregular)
18		CA—Constructive	38		CA—Constructive
19		CAA—Constructive-M. Area	39		CAA—Constructive-M. Area
20		CAD—Constructive-M. Adjacency	40		CAD—Constructive-M. Adjacency

where Ap represents the item area; Ao the object area; and n the number of items inside the object. Then, the fitness is given by

$$FF = \frac{\sum_{u=1}^{No} P_u^2}{No} \tag{2}$$

where No is the total number of objects used, and P_u is the fractional utilization for each object u. We wish to maximise FF; note that each $P_u \leq 1$. Now, how is the fitness of a chromosome measured during the GA cycle? To do this, first, it is necessary to compute the fitness produced by each individual combination of selection and placement heuristics, for each instance. The best heuristic combination and its result, for each specified instance i are stored (let us call it BSH_i). These results are prepared before running the GA. The steps of the GA cycle are:

1. Generate initial population.
2. Assign 5 problems to each chromosome and compute its fitness.
3. Apply selection (tournament), crossover and mutation operators to produce two children.
4. Assign 5 problems to each new child and obtain its fitness.
5. Replace the two worst individuals 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 cycle), 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:

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

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 (2)).

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

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

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 the fitness function given by (2).

Genetic operators. In this investigation we used two crossover and three mutation operators. The first crossover operator is very similar to the normal two-point crossover, but where the points selected inside each corresponding block are the same for both parents. Since the number of blocks in each chromosome is variable, the cut points in each parent are chosen independently. However the points selected inside each corresponding block are forced to be the same for both parents, so that the recombination produces an exact number of blocks. The blocks and points 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 from 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 that value with a new number between -3 and 3, generated with a normal distribution with mean 0.5 and truncated accordingly.

4 Experiments and results

This section presents the experiments carried out during the investigation and the results obtained. We begin with results for rectangular instances. Then, we report results for the case in which the problems consist of packing polygonal pieces in rectangular objects.

4.1 2D Regular Packing: experimenting with single heuristics

Problems from several sources have been used in this research. Part of the benchmark set was taken from the literature (the OR-Library (Beasley 2003), a set by Martello and Vigo (1998), a set by Berkey and Wang (1987), a set by Terashima-Marín et al. (2005a)), and the rest is composed of randomly generated instances for which an optimal solution is known.

Table 3 Single heuristics: number of extra objects for problems in set A

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

Table 4 Single heuristics: number of extra objects for problems in set B

Obj	FF	FFD	FFI	Filler	NF	NFD	BF	BFD	WF	DJD
	1–4	5–8	9–12	13–16	17–20	21–24	25–28	29–32	33–36	37–40
0	28.49	65.32	13.44	68.28	29.84	63.84	29.3	66.67	28.76	**71.77**
1	32.80	31.45	21.77	28.49	30.91	25.13	31.18	30.11	32.53	**25.27**
2	20.97	2.96	8.60	2.96	19.35	2.02	20.7	2.96	20.96	**2.69**
3	9.68	0	11.02	0	11.56	0	10.57	0	9.41	**0**
4	6.72	0.27	8.60	0.27	6.99	0.13	6.72	0.27	6.99	**0.27**
>4	1.34		36.56		1.34		1.34		1.34	

The collection includes 1080 different instances. They were divided into two groups (A and B) of 540 instances each (chosen at random). To test the model effectiveness, various kinds of experiments were carried out.

All combinations of single heuristics were tested with the two instance groups. This was done with the purpose to establish the basis for comparison on the performance of the hyper-heuristics delivered by the proposed model. The results of the single heuristics in Group A are shown in Table 3. Table 4 shows the results when the single heuristics where tested with Group B.

Results are grouped in sets of four heuristics in which the selection heuristic is the same for all members of any given set. For example, column 2 of Table 3 indicates results for the first four combinations in which the common selection heuristic is First Fit. Results are compared against those generated by the best single heuristics. Figures in cells indicate the percentage of problems solved with a particular number of extra objects (left column) when compared with results provided by the best heuristic. For example, in the column labeled 'FFD', heuristics 5 to 8 solved 70.43% (this is the average performance on the four heuristics) of the instances in set A with the same number of objects as the best heuristic for each given instance. In both Tables (3 and 4) the best results are provided by the set whose common selection heuristic is the DJD (figures in bold). This is not unexpected since the DJD heuristic is, in general, a very good heuristic for bin-packing. There are other sets of heuristics showing also a very good performance (13–16, 29–32) in which the selection heuristic is the Filler plus First Fit, and Best Fit Decreasing, respectively.

4.2 2D Regular Packing: experimenting with the proposed model

To produce the hyper-heuristics, the proposed model was tested as follows. Five complete runs (Roman numerals I to V) were done, where a single complete run starts from scratch from the training phase. Then for each run, and after the training phase was finished (that is, using set A only), the best five hyper-heuristics produced were taken, tested and compared using the two instance sets (separately), and then the best one was chosen for the summarized results (reported below in Table 10).

Run I produced results shown in Table 5. The left portion shows results with the best five hyper-heuristics for the training set A, and the right portion shows results for the testing set B. That means, for example, that I-1A is the hyper-heuristic obtained with set A, and the corresponding results when that hyper-heuristic was tested with set B are shown in column labeled I-1B. The best five hyper-heuristics produced by that run solve on average 88.6% ($1.61 + 94.09 + 1.61 + 89.78 + 1.08 + 92.47 + 0 + 81.18 + 0 + 81.18$ divided by 5) of problems in set A with at the same or fewer numbers of objects as the best single heuristic. Performance on set B is quite similar, confirming the effectiveness of this model. Tables 6, 7, 8 and 9 show results for each of the remaining runs. It is interesting to observe that the hyper-heuristics produced in these runs deliver results in which the number of objects is 2 less than those produced by the best single heuristics. This happens for the testing set B.

Table 10 summarizes results for all five runs. The best hyper-heuristic for each run was selected and kept in the left columns (second to sixth). The corresponding results for those hyper-heuristics on set B are presented in the right part of the Table (columns seven to eleven). The average performance of these heuristics on set A is around 89.46% considering only results with the same or fewer (-1) objects compared to single heuristics. The average performance of the same heuristics, but for set B, lies around 85.6%. It is again interesting to observe in these results that, for set B, there are some hyper-heuristics that obtain some solutions using two fewer objects.

Table 5 Results on Run 1 with training and testing sets A and B

Obj	I-1A	I-2A	I-3A	I-4A	I-5A	I-1B	I-2B	I-3B	I-4B	I-5B
−1	1.61	1.61	1.08	0	0	2.15	2.15	2.69	1.08	1.61
0	94.09	89.78	92.47	81.18	81.18	81.18	76.88	80.11	71.51	73.12
1	4.30	8.60	6.45	18.82	18.82	16.67	20.97	17.20	27.42	25.27
>2	0	0	0	0	0	0	0	0	0	0

Table 6 Results on Run 2 with training and testing sets A and B

Obj	I-1A	I-2A	I-3A	I-4A	I-5A	I-1B	I-2B	I-3B	I-4B	I-5B
−2	0	0	0	0	0	0.54	0.54	0.54	0.54	0.54
−1	3.23	2.69	2.69	2.15	2.69	8.6	7.53	5.38	4.84	8.60
0	76.88	77.42	76.88	72.58	75.81	72.58	72.58	73.12	73.12	74.19
1	19.89	19.89	20.43	25.27	21.51	18.28	19.35	20.97	21.51	17.21
>2	0	0	0	0	0	0	0	0	0	0

Table 7 Results on Run 3 with training and testing sets A and B

Obj	I-1A	I-2A	I-3A	I-4A	I-5A	I-1B	I-2B	I-3B	I-4B	I-5B
−2	0	0	0	0	0	0.54	0.54	0.54	0.54	0.54
−1	5.38	4.30	5.38	4.30	4.30	4.84	4.84	8.06	3.76	8.60
0	83.33	76.34	84.41	72.58	81.18	73.12	73.12	77.96	71.51	76.88
1	11.29	19.35	10.22	22.58	14.52	21.51	21.51	13.44	24.19	13.98
>2	0	0	0	0	0	0	0	0	0	0

Table 8 Results on Run 4 with training and testing sets A and B

Obj	I-1A	I-2A	I-3A	I-4A	I-5A	I-1B	I-2B	I-3B	I-4B	I-5B
−2	0	0	0	0	0	1.08	1.08	1.08	1.08	1.08
−1	6.45	5.38	6.45	4.84	4.30	5.38	4.84	7.53	3.23	8.06
0	84.95	78.49	86.56	82.8	83.33	82.8	80.65	80.11	76.34	76.88
1	8.60	16.13	6.99	11.83	12.37	10.57	13.44	11.29	19.35	13.98
> 2	0	0	0	0	0	0	0	0	0	0

Table 9 Results on Run 5 with training and testing sets A and B

Obj	I-1A	I-2A	I-3A	I-4A	I-5A	I-1B	I-2B	I-3B	I-4B	I-5B
−2	0	0	0	0	0	10.54	0.54	0.54	0.54	0.54
−1	3.76	3.23	4.3	3.23	3.23	10.22	9.68	9.14	7.53	8.60
0	87.63	86.56	83.87	80.11	76.34	80.65	80.11	79.03	75.27	73.66
1	8.60	10.22	11.83	16.67	20.43	8.6	9.68	11.29	16.67	17.20
> 2	0	0	0	0	0	0	0	0	0	0

Table 10 The best five hyper-heuristics for all runs and test sets A and B

Obj	I-A	II-A	III-A	IV-A	V-A	I-B	II-B	III-B	IV-B	V-B
−2	0	0	0	0	0	0	0.54	0.54	1.08	0.54
−1	1.61	3.23	5.38	6.45	3.76	2.69	8.6	8.6	8.06	10.22
0	94.09	76.88	83.33	84.95	87.63	80.11	72.58	76.88	76.88	80.65
1	4.3	19.89	11.29	8.60	8.60	17.2	18.28	13.98	13.98	8.60

4.3 2D Irregular Packing: generating irregular instances

We designed a special algorithm to randomly produce problem instances containing irregular pieces (actually it is a modification to the algorithm used to produce random instances with rectangular pieces). The pieces generated are convex polygons with a number of sides between 3 and 8. The algorithm starts by generating in a random fashion an initial number of rectangles, specified by a given parameter. The next step is to divide up the rectangles into a number of pieces, until the total number of pieces is completed. The parameters needed to create an instance are the number of objects, their dimensions, the number of pieces in

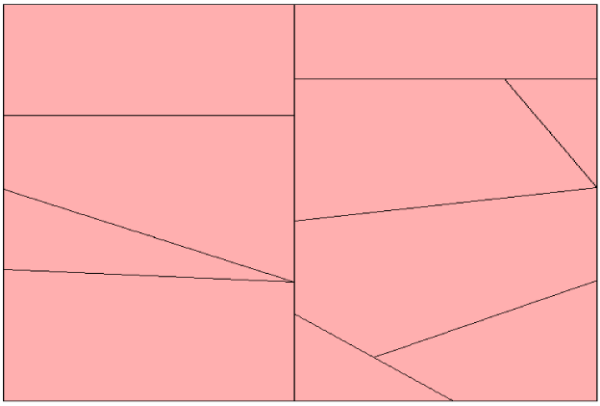

Fig. 7 Example of an irregular instance

each object, the minimum side in a piece, the maximum ratio between the largest and smallest side (it determines the irregularity or rectangularity factor), and the initial number of rectangles. Figure 7 shows an example of an instance with one object and 10 pieces.

For the irregular packing problem in particular, we generated 540 instances contained within 18 different types. 30 instances were generated for each type. Their characteristics such as object size, number of pieces, number of objects can be seen in Table 11. We also added a problem from the literature (Fujita et al. 1993). The instance was scaled by a factor of 10 in order to have the object size of 300×300.

Instances of type G are the only problems with unknown optimal number of objects and this is due to the fact that they were produced after alterations to randomly generated problems with known optimum. The alteration consisted of randomly eliminating 9 pieces from the original instance containing 45 pieces. All problems present the same number of pieces per object, except for types C, K, P and R. Problems type C have three objects with four pieces each, and the remaining three objects with eight pieces each. Instances of type K have three objects with three pieces each, and three objects with fifteen pieces each. This kind of problem has some pieces of around a third of the object size, but also a large number of small pieces. The type P problems have 2 objects with 2 pieces each, two more objects with five pieces each, 2 objects with 9 pieces each, and two additional objects with 12 pieces each. Problem type R has nine objects with 2, 3, ... and 10 pieces respectively.

In general, an instance can be considered *more* irregular (irregularity factor) if it contains pieces with a very small minimum side, and the ratio (*largest side*)/(*smallest side*) rather large producing pieces with small rectangularity factor. The rectangularity factor refers to the proportion between the area of a piece and the area of a horizontal rectangle containing it. The combination of pieces with small minimum side and larger maximum ratio (appearing in problem types L, M, and O), produces many very narrow and elongated pieces, many of which run across the object from one side to the other. The greater the initial number of rectangles (it is an input parameter for the algorithm), the more rectangles or pieces with horizontal or vertical sides the instance tends to have. These kinds of instances are somehow easier to solve since it is more likely to find a match with the object edge, or with other pieces with the same kind of sides. The instances in problem type I contain only rectangles. Table 12 shows the irregularity factor by problem type.

Table 11 Description of problem instances

Type	Objects	Pieces	Number of instances	Optimal (number of objects)
Fu	300 × 300	12	1	unknown
Type A	1000 × 1000	30	30	3
Type B	1000 × 1000	30	30	10
Type C	1000 × 1000	36	30	6
Type D	1000 × 1000	60	30	3
Type E	1000 × 1000	60	30	3
Type F	1000 × 1000	30	30	2
Type G	1000 × 1000	36	30	unknown
Type H	1000 × 1000	36	30	12
Type I	1000 × 1000	60	30	3
Type J	1000 × 1000	60	30	4
Type K	1000 × 1000	54	30	6
Type L	1000 × 1000	30	30	3
Type M	1000 × 1000	40	30	5
Type N	1000 × 1000	60	30	2
Type O	1000 × 1000	28	30	7
Type P	1000 × 1000	56	30	8
Type Q	1000 × 1000	60	30	15
Type R	1000 × 1000	54	30	9
Total			541	

4.4 2D Irregular Packing: experimenting with proposed model

In order to test the overall performance of the model when tackling irregular problems, various experiments were designed and they are presented in Table 13. The experiments are described as follows:

- **Experiment Type I**. First, instances are divided into two groups: Group A and Group B. Group A is the training set and is formed of instance types A, B, C, D, E, F, G, H and I plus the *Fu* instance totaling 271 instances. The best individual generated by the evolving process within the proposed model is the hyper-heuristic chosen to run the rest of the instances comprising Group B (instance type J, K, L, M, N, O, P, Q and R), which is the testing set. To validate the consistency of the model, two complete and independent runs were performed, from which hyper-heuristics HH1a and HH2b were obtained. Each was tested with Set B and then compared with those results obtained by each of the 40 single heuristics (each is a combination of selection and placement heuristic) in the same set.
- **Experiment Type II**. This experiment is similar to Experiment Type I, except that the training and testing sets are interchanged.
- **Experiment Type III**. This experiment takes the *Fu* instance and 15 instances from each problem type (from A through R) to form the training set with 271 instances. The remaining instances form the testing set.
- **Experiment Type IV**. It is the same as Experiment Type III, except that the training and testing sets are swapped.

Table 12 Irregularity in the generated problems

Type	Minimum side	Maximum ratio (*largest side*)/(*minimum side*)	Initial rectangles per object
Type A	100	3	3
Type B	100	3	1
Type C	100	3	1
Type D	100	3	3
Type E	50	10	1
Type F	50	5	3
Type G	100	3	1
Type H	100	3	1
Type I	100	3	19
Type J	100	3	9
Type K	100	4	1
Type L	10	10	1
Type M	10	10	1
Type N	30	5	8
Type O	10	10	0
Type P	25	5	0
Type Q	200	2	1
Type R	80	4	1

Table 13 Description of the experiments

Experiment	Training set	Testing set	Hyper-heuristic
I	Instance *Fu* and problems Type A to I (270 problems)	Problems Type J to R (271 problems)	HH1a HH1b
II	Problems Type J to R (270 problems)	Instance *Fu* and problems Type A to I (271 problems)	HH2
III	Instance *Fu* and 15 first problems from each Type A to R (271 problems)	Last 15 problems from each Type A to R (270 problems)	HH3
IV	Last 15 problems from each Type A to R (270 problems)	Instance *Fu* and 15 first problems from each Type A to R (271 problems)	HH4

All problem instances (541) were solved with the 40 combinations of selection and placement heuristics. For each instance, the best heuristic, its fitness and the number of objects are recorded. In many cases, several heuristics provide similar results with respect to the number of objects, but the one with highest fitness is taken. Heuristic 7 (FFD + Constructive Approach-Minimum Area) appeared to be the best in 192 instances (35.5%). Heuristic 15 (Filler + FFD + Constructive Approach-Minimum Area) was the best in 113 instances (20.9%).

The resulting hyper-heuristics in Experiment Type I show better results than those produced by the single heuristics. In general, the training instances in this experiment seem to

Table 14 Number of extra objects for the testing set and compared against results of the best single heuristics. Experiment I, for first and second independent runs

Obj	HHa	HHb	FF 1–4	FFD 5–8	FFI 9–12	Filler 13–16	NF 17–20	NFD 21–24	BF 25–28	BFD 29–32	WF 33–36	DJD 37–40
0	87.0	90.4	5.0	54.4	0.5	55.6	3.7	52.7	10.5	55.0	10.5	30.9
1	13.0	9.6	32.4	22.2	20.7	21.9	29.5	23.5	34.4	21.9	34.4	32.3
2			28.1	11.4	20.2	11.0	27.4	11.3	25.4	11.1	25.4	17.3
3			15.6	5.8	18.9	5.6	16.7	6.1	13.1	5.8	13.1	10.1
4			10.5	3.7	14.6	3.8	11.9	3.4	8.3	3.9	8.3	4.3
5			4.0	2.0	9.7	1.9	5.5	2.5	3.8	1.9	3.8	3.5
>5			4.4	0.4	15.4	0.3	5.3	0.5	4.5	0.3	4.5	1.6

Table 15 Number of extra objects for the testing set and compared against results of the best single heuristics. Experiment II

Obj	HH	FF 1–4	FFD 5–8	FFI 9–12	Filler 13–16	NF 17–20	NFD 21–24	BF 25–28	BFD 29–32	WF 33–36	DJD 37–40
−1	0.4										
0	86.7	17.2	57.5	8.8	58.4	16.4	55.9	24.2	58.4	24.2	38.7
1	10.0	39.2	26.3	27.2	25.6	36.9	26.8	38.7	25.7	38.7	28.2
2	2.6	22.3	9.5	24.2	9.9	20.8	10.0	22.2	9.7	22.2	15.7
3	0.4	13.6	4.0	14.9	3.6	16.1	4.1	9.0	3.3	9.0	10.2
4		5.7	2.0	14.8	1.8	6.4	2.4	4.6	2.1	4.6	5.2
5		1.7	0.6	8.7	0.6	2.7	0.7	1.0	0.6	1.0	1.8
>5		0.4	0.1	1.5	0.1	0.6	0.1	0.3	0.1	0.3	0.2

be less irregular than those in the testing set (see Table 12). Table 14 presents results for the two hyper-heuristics produced (HHa and HHb) considering only the testing set. Hyper-heuristic HHa solved 87.0% with the same number of objects as the best single heuristic. In the remaining 13% of the instances the hyper-heuristic required just one additional object. Figures in columns 3 to 12 average the performance of the four combinations of heuristics for which the selection heuristic is in common. The closest combination is 'Filler' with 55.6%. Moreover, in a good percentage of cases all the single heuristics use more than one object, and in fact, all of them have instances using more than five additional objects. The performance in both hyper-heuristics is very similar, despite the fact that they are formed of rather different blocks. We ran χ^2 tests to verify if there are statistically significant differences between both hyper-heuristics. Results indicate that the observed differences come from causes related to randomness ($p - value = 0.853$). This is a good indication that the method to build the hyper-heuristics is robust enough to generate hyper-heuristics very similar in performance.

Experiment Type II is the opposite to Experiment Type I. In other words, the training set in this experiment is composed of more irregular pieces, and then tested with less irregular pieces. Table 15 shows results for this experiment. It can be observed that the hyper-heuristic

Table 16 Number of extra objects for the testing set and compared against results of the best single heuristics. Experiment III

Obj	HH	FF	FFD	FFI	Filler	NF	NFD	BF	BFD	WF	DJD
		1–4	5–8	9–12	13–16	17–20	21–24	25–28	29–32	33–36	37–40
0	91.1	10.8	55.4	4.5	56.4	9.7	53.7	18.1	55.9	18.1	35.2
1	7.8	37.7	24.8	23.8	24.5	34.9	25.1	35.1	24.3	35.1	27.9
2	1.1	24.2	10.4	23.2	10.3	22.8	11.0	24.9	10.8	24.9	17.6
3		15.5	5.5	17.4	5.0	17.7	6.0	11.3	4.7	11.3	11.3
4		7.5	2.4	13.8	2.4	8.7	2.0	5.9	2.8	5.9	4.8
5		2.1	1.3	8.9	1.2	3.6	1.9	2.5	1.3	2.5	2.4
>5		2.2	0.3	8.3	0.2	2.6	0.3	2.1	0.2	2.1	0.8

Table 17 Number of extra objects for the testing set and compared against the best results of the best single heuristics. Experiment IV

Obj	HH	FF	FFD	FFI	Filler	NF	NFD	BF	BFD	WF	DJD
		1–4	5–8	9–12	13–16	17–20	21–24	25–28	29–32	33–36	37–40
0	87.8	11.3	56.5	4.7	57.7	10.4	54.9	16.5	57.5	16.5	34.5
1	10.7	33.9	23.7	24.2	22.9	31.5	25.3	38.0	23.4	38.0	32.7
2	1.1	26.2	10.5	21.1	10.6	25.5	10.2	22.7	10.0	22.7	15.4
3		13.7	4.3	16.4	4.2	15.1	4.2	10.8	4.4	10.8	9.0
4		8.7	3.3	15.6	3.2	9.6	3.8	7.0	3.2	7.0	4.6
5		3.5	1.4	9.5	1.3	4.5	1.4	2.3	1.3	2.3	2.9
>5		2.6	0.2	8.5	0.2	3.3	0.3	2.7	0.2	2.7	0.9

produced solves 86.7% of the instances with the same number of objects or fewer compared to the best heuristic. For 0.4% of the cases it needs one object less. For around 13%, it needs more than one object and up to three. The combinations of single heuristics with 'Filler' as the selection heuristic is the one obtaining the best results with 58.4%. Again, all single heuristics require more than five objects for some instances.

For Experiments III and IV, the training and testing sets had similarities in their instances given that the problem types were evenly divided, leaving half of the instances in the training set, and the other half in the testing set. This explains, in average, a very similar performance in the hyper-heuristics produced by these experiments. The results are shown in Tables 16 and 17. For Experiment III the hyper-heuristic produces 91.1% with no additional objects, 7.8% with one additional object and just 1.1% with two. The best single heuristics are again the 'Filler'-based, but the 'FFD's and 'BFD's are also close. For Experiment IV, performance of the hyper-heuristic is very similar reaching 87.8% of the instances with no additional objects.

Table 18 Number of extra objects used by the hyper-heuristics in the testing set with respect to the average obtained by the 40 single heuristics

Extra objects	
Experiment I, 1st	−1.63
Experiment I, 2nd	−1.66
Experiment II	−1.11
Experiment III	−1.41
Experiment IV	−1.37

4.5 2D Irregular Packing: analysis on the hyper-heuristics produced

Table 18 summarizes with a single value, the performance of each hyper-heuristic for each experiment carried out. On average, using the hyper-heuristic reduces by more than one object the solution given by the best heuristic for each instance. The hyper-heuristics produced in Experiment I were the ones showing the best performance. Results for Experiment II are not as good as the previous experiment. It seems that training with irregular instances first, and then testing with *more* rectangular problems is not so advantageous.

5 General discussion

Looking at the results it is clear in all cases that the method to create 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, the computational cost of applying a generated hyper-heuristic to a problem is only slightly higher than the time used by any of the simple heuristics, which run in just a few seconds.

When solving the 2D irregular bin-packing problem, the experiments generated five hyper-heuristics. It is interesting to observe a slightly better performance when the hyper-heuristics are tested with more irregular instances. The computation time to solve the 541 instances with the 40 single heuristics was around 12 hours. We keep track of the results for each instance for each single combination of heuristics. Every time the training and testing sets are redefined, it is necessary to repeat the process again. Running the GA to generate a hyper-heuristic takes around 11 hours. Table 19 shows the time taken in each experiment, and there is, as expected, an obvious correspondence between the time taken and the size of the problems, related to the number of pieces. There is a correlation index of around 0.95 between the size of the training instances in the training phase and the time taken to generate the hyper-heuristic. Had the size of the instance been measured as the number of objects in the optimal solution, a similar relationship would have been observed.

Solving each instance of the testing set with the produced hyper-heuristic is considerably faster. This phase takes just few minutes to process all instances. Solving an instance with the hyper-heuristic is faster than solving the instance with each of the 40 single heuristics and then choosing the best result. On average, we could say that the hyper-heuristic solves an instance in about $1/40$ of the time that would be needed to find the best result of all the single heuristics.

Table 19 2D Irregular: Time required to generate each hyper-heuristic

Experiment	Elapsed time (hours)	Instance average size (Training) (number of pieces)
Experiment I, 1st	9.1	42.0
Experiment I, 2nd	9.5	42.0
Experiment II	13.1	49.1
Experiment III	10.8	45.6
Experiment IV	12.2	45.6

6 Conclusions and future work

This paper has described experimental results of 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 and Irregular bin-packing problems. These combinations are called hyper-heuristics. Overall, the scheme efficiently identifies 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, with arbitrary shapes. It would be also interesting to extend the approach to other multidimensional problems.

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

References

Bai, R., Burke, E. K., & Kendall, G. (2008). Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. *Journal of the Operational Research Society* (to appear).

Banzhaf, W., Nordin, P., Keller, R. E., & Francone, F. D. (1998). *Genetic programming: an introduction*. London: Morgan Kaufmann.

Beasley, J. E. (2003). Operations research library. Collection of problems for 2D packing and cutting. http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

Berkey, J. O., & Wang, P. Y. (1987). Two-dimensional finite bin packing algorithms. *Journal of Operational Research Society*, *38*(5), 423–429.

Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., & Schulenburg, S. (2003a). Hyper-heuristics: an emerging direction in modern research technology. In *Handbook of metaheuristics* (pp. 457–474). Dordrecht: Kluwer Academic.

Burke, E. K., Kendall, G., & Soubeiga, E. (2003b). A tabu-search hyperheuistic for timetabling and rostering. *Journal of Heuristics*, *9*(6), 451–470.

Burke, E. K., Hyde, M. R., & Kendall, G. (2006). Evolving bin packing heuristics with genetic programming. In *9th PPSN* (pp. 860–869). Reykjavik. LNCS.

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007). A graph-based hyperheuristic for timetabling problems. *European Journal of the Operational Research*, *176*(1), 177–192.

Cheng, C. H., Fiering, B. R., & Chang, T. C. (1994). The cutting stock problem. A survey. *International Journal of Production Economics*, *36*, 291–305.

Dowsland, K. A., Dowsland, W. B., & Bennell, J. A. (1998). Jostling for position: local improvement for irregular cutting patterns. *Journal of the Operational Research Society*, *49*(6), 647–658.

Dowsland, K. A., Vaid, S., & Dowsland, W. B. (2002). An algorithm for polygon placement using a bottom-left strategy. *European Journal of Operational Research*, *141*(2), 371–381.

Dowsland, K., Herbert, E., Kendall, G., & Burke, E. (2006). Using the search bounds to enhance a genetic algorithms approach to two rectangle packing problems. *European Journal of Operational Research*, *168*(2), 390–402.

Dowsland, K., Soubeiga, E., & Burke, E. K. (2007). A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of the Operational Research*, *179*(3), 759–774.

Dyckhoff, H. (1990). A topology of cutting and packing problems. *European Journal of Operational Research*, *44*, 145–159.

Fogel, D. B., Owens, L. A., & Walsh, M. (1966). *Artificial intelligence through simulated evolution*. New York: Wiley.

Fujita, K., Akagji, S., & Kirokawa, N. (1993). Hybrid approach for optimal nesting using a genetic algorithm and a local minimisation algorithm. In *Proceedings of the 19th annual ASME design automation conference, Part 1 (of 2)* (Vol. 65, part 1, pp. 477–484). Albuquerque, NM, USA.

Garey, M., & Johnson, D. (1979). *Computers and intractability*. New York: W. H. Freeman.

Goldberg, D. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading: Addison-Wesley.

Goldberg, D., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, *3*, 493–530.

Golden, B. L. (1976). Approaches to the cutting stock problem. *AIIE Transactions*, *8*, 256–274.

Hifi, M., & MHallah, R. (2002). A best-local position procedure-based heuristic for two-dimensional layout problems. *Studia Informatica Universalis, International Journal on Informatics*, *2*(1), 33–56.

Hifi, M., & MHallah, R. (2003). A hybrid algorithm for the two-dimensional layout problem: the cases of regular and irregular shapes. *International Transactions in Operational Research*, *10*, 195–216.

Holland, J. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.

Hopper, E., & Turton, B. C. (2001a). An empirical investigation of metaheuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, *128*(1), 34–57.

Hopper, E., & Turton, B. C. (2001b). An empirical study of meta-heuristics applied to 2D rectangular bin packing. *Studia Informatica Universalis*, *2*(1), 77–106.

Jakobs, S. (1996). On genetic algorithms for the packing of polygons. *European Journal of Operations Research*, *88*, 165–181.

Kantorovich, L. V. (1960). Mathematical methods of organizing and planning production. *Management Science*, *6*, 366–422.

Kendall, G., Soubeiga, E., & Cowling, P. (2004). Choice function and random hyperheuristics. In N. Press (Ed.), *4th Asia-Pacific conference on simulated evolution and learning* (pp. 667–671). Nanyang.

Kröger, B. (1995). Guillotineable bin packing: A genetic approach. *European Journal of Operational Research*, *84*(2), 645–661.

Liu, D., & Teng, H. (1999). An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangle. *European Journal of Operations Research*, *112*, 413–419.

Marín-Blázquez, J. G., & Schulenburg, S. (2007). A hyper-heuristic framework for XCS: learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In P. L. Lanzi, W. Stolzmann, & S. W. Wilson (Eds.), *Learning classifier systems* (pp. 193–218). Berlin: Springer.

Martello, S., & Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management Science*, *44*(3), 388–399.

Poli, R., Woodward, J., & Burke, E. K. (2007). A histogram-matching approach to the evolution of bin packing strategies. In *Proceedings of congress on evolutionary computation CEC2007* (pp. 3500–3507). Singapore.

Rechenberg, I. (1973). *Evolutionstrategie: optimierung technischer systeme nach prinzipien dier biolischen evolution*. Stuttgart: Frommann-Holzboog.

Reeves, C. (1996). Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, *63*(3), 371–396.

Ross, P., Schulenburg, S., Blázquez, J. M., & Hart, E. (2002). Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In *Proceedings of GECCO 2002* (pp. 942–948).

Ross, P., Blázquez, J. M., Schulenburg, S., & Hart, E. (2003). Learning a procedure that can solve hard bin-packing problems: a new GA-based approach to hyper-heuristics. In *Proceedings of GECCO 2003* (pp. 1295–1306).

Schwefel, H. P. (1981). *Numerical optimization of computer models*. Chichester: Wiley.

Terashima-Marín, H., Flores-Álvarez, E. J., & Ross, P. (2005a). Hyper-heuristics and classifier systems for solving 2D-regular cutting stock problems. In *Proceedings of the genetic and evolutionary computation conference 2005* (pp. 637–643).

Terashima-Marín, H., Morán-Saavedra, A., & Ross, P. (2005b). Forming hyper-heuristics with GAs when solving 2D-regular cutting stock problems. In *Proceedings of the congress on evolutionary computation* (pp. 1104–1110), 2005.

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. In *Proceedings of the genetic and evolutionary computation conference 2006* (pp. 591–598).

Uday, A., Goodman, E. D., & Debnath, A. A. (2001). Nesting of irregular shapes using feature matching and parallel genetic algorithms. In E.D. Goodman (Ed.), *2001 genetic and evolutionary computation conference late breaking papers* (pp. 429–434). San Francisco, California, USA.

Wilson, R. A., & Keil, F. C. (1999). *The MIT encyclopedia of the cognitive science*. Cambridge: MIT Press.

Wäscher, G., Haussner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, *183*(3), 1109–1130.