Expert Systems with Applications 41 (2014) 6876–6889

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A unified hyper-heuristic framework for solving bin packing problems

Eunice López-Camacho [a], Hugo Terashima-Marin [a,*], Peter Ross [b], Gabriela Ochoa [c]

[a] *Tecnológico de Monterrey, Av. E. Garza Sada 2501, Monterrey, NL 64849, Mexico*
[b] *School of Computing, Edinburgh Napier University, Edinburgh EH10 5DT, UK*
[c] *Computing Science and Mathematics, University of Stirling, Scotland, UK*

A R T I C L E I N F O

Article history:
Available online 10 May 2014

Keywords:
Bin packing problems
Evolutionary computation
Hyper-heuristics
Heuristics
Optimization

A B S T R A C T

One- and two-dimensional packing and cutting problems occur in many commercial contexts, and it is often important to be able to get good-quality solutions quickly. Fairly simple deterministic heuristics are often used for this purpose, but such heuristics typically find excellent solutions for some problems and only mediocre ones for others. Trying several different heuristics on a problem adds to the cost. This paper describes a hyper-heuristic methodology that can generate a fast, deterministic algorithm capable of producing results comparable to that of using the best problem-specific heuristic, and sometimes even better, but without the cost of trying all the heuristics. The generated algorithm handles both one- and two-dimensional problems, including two-dimensional problems that involve irregular concave polygons. The approach is validated using a large set of 1417 such problems, including a new benchmark set of 480 problems that include concave polygons.

1. Introduction

Finding an arrangement of pieces to cut or pack inside larger objects is known as the cutting and packing problem. Besides the academic interest in this NP-hard problem, there are numerous industrial applications of its many variants. The one-dimensional (1D) and two-dimensional (2D) bin packing problems (BPPs) are particular cases of the cutting and packing problem. The 1D BPP can be applied, for example, to the assignment of commercial breaks on television and for copying a collection of files to disks (Bhatia, Hazra, & Basu, 2009). For the 2D BPP, the case of rectangular pieces is the most widely studied. However, the irregular case is seen in a number of industries where parts with irregular shapes are cut from rectangular materials. For instance, in the shipbuilding industry, plate parts with free-form shapes for use in the inner frameworks of ships are cut from rectangular steel plates, and in the garment industry, parts of clothes and shoes are cut from fabric or leather (Okano, 2002). Other applications include the optimization of layouts within the wood, sheet metal, plastics, and glass industries (Burke, Hellier, Kendall, & Whitwell, 2006). In these industries, improvements of the arrangement can result in a large saving of material (Hu-yao & Yuan-jun, 2006).

Hyper-heuristics aim at automating the design of heuristic methods to solve difficult search problems and providing a more general procedure for optimization (Burke et al., 2003; Pillay, 2012; Burke, Gendreau, Hyde, Kendall, & Ochoa, 2013). Hyper-heuristics differ from the widely-used term meta-heuristic: instead of searching within the space of solutions, they explore the space of heuristics (Vázquez-Rodríguez, Petrovic, & Salhi, 2007; Pappa et al., 2013). The idea is to use a variety of methods to discover algorithms, based on single heuristics, that have good worst-case performance across a range of problems and are fast in execution (Ross, 2014). There are two main types of hyper-heuristic: selection hyper-heuristics, which are methods for choosing or selecting existing heuristics, and generation hyper-heuristics which focus on generating new heuristics from components of existing heuristics (Burke et al., 2013; Burke et al., 2010a). The approach presented in this paper is of the first type.

Over the last few years, one trend in combinatorial optimization has been to find more general solvers capable of extending to other types of problems within a domain and even crossing domain boundaries. For example, Burke et al. (2010b) conducted an empirical study that ran the same hyper-heuristic strategy in three different domains: 1D bin packing, permutation flow shop and personnel scheduling. Burke, Hyde, Kendall, and Woodward (2012) presented a genetic programming system to automatically generate a good quality heuristic for each instance of the one-, two-, and three-dimensional knapsack and bin packing problems with rectilinear pieces; however, because the generated heuristics

* Corresponding author. Tel.: +52 8181582045.
E-mail addresses: eunice.lopez@itesm.mx (E. López-Camacho), terashima@itesm.mx (H. Terashima-Marin), P.Ross@napier.ac.uk (P. Ross), gabriela.ochoa@cs.stir.ac.uk (G. Ochoa).

http://dx.doi.org/10.1016/j.eswa.2014.04.043
0957-4174/© 2014 Elsevier Ltd. All rights reserved.

are instance-specific, the computational costs involved are non-trivial. Ochoa et al. (2012) proposed a software framework named HyFlex (Hyper-heuristic Flexible framework) for developing cross-domain search methodologies along six different optimization problems.

In this paper, we introduce an evolutionary hyper-heuristic framework for solving 1D and 2D BPPs (rectangular, convex and concave shapes) that automatically chooses which heuristic to apply at each step in building a good solution. The approach described in this paper is a development of earlier work on solving the 1D BPP (Ross, Marín-Blázquez, Schulenburg, & Hart, 2003), the 2D regular packing problem (Terashima-Marín, Farías-Zárate, Ross, & Valenzuela-Rendón, 2006) and the 2D irregular (convex only) packing problem (Terashima-Marín, Ross, Farías-Zárate, López-Camacho, & Valenzuela-Rendón, 2010). In that earlier work, the solution construction process used an ad-hoc simplification of the current problem state when deciding what to do next and, in the 2D cases, a large set of possible basic heuristics.

The main contributions of this paper are:

- A unified framework that handles 1D, 2D regular (rectangles), and 2D irregular (convex and non-convex polygons) packing problems, together with an empirical analysis of its performance on a large unseen set of such problems.
- An experiment-based methodology for deciding which heuristics should be included in the framework.
- A data-mining methodology for choosing the problem-state representation to be used.
- The creation of a new, large benchmark set of 2D problems that include some non-convex polygons.

2. Background and related work

Many heuristics have been developed for specific problems but none of them seems able to provide good-quality results for all instances. Certain problems may contain features that enable a particular heuristic to work well, but those features may not be present in other problems and so might lower that heuristic's performance. Research in hyper-heuristics has developed algorithms with some claims to more generality, but there is interest in seeing whether even more general architectures can be developed, that are capable of solving many different kinds of problem efficiently. Recent work by Ochoa et al. (2012), introduced a software framework called HyFlex for the development of cross-domain search methodologies. The framework provides a common interface for treating different combinatorial problems and provides the problem-specific algorithm components. Hyflex can be seen as a benchmark framework for developing, testing and comparing the generality of algorithms such as selection hyper-heuristics. HyFlex has served to test algorithms in different domains like maximum satisfiability, one dimensional bin packing, permutation flow shop, personnel scheduling, traveling salesman and vehicle routing. Other interesting investigations have been motivated by the HyFlex system, see for example the work by Burke et al. (2010b) where several hyper-heuristics combining two heuristic selection and three acceptance approaches were compared, and other extensions are given in Burke, Gendreau, Ochoa, and Walker (2011). In a related study, Burke et al. (2012) proposed a general packing methodology that includes 1D, 2D (orthogonal) and 3D (orthogonal) bin packing and knapsack packing. They presented a genetic programming system to automatically generate a good quality heuristic for each instance among the different problems considered although at a non-trivial cost per instance. HyFlex has also served as a framework for the CHeSC 2011 algorithm competition, won by Misir, Verbeeck, Causmaecker, and Berghe (2011) with an algorithm which provides an intelligent way of selecting heuristics, pairing heuristics and adapting the parameters of heuristics online. They later extended this (Misir, Verbeeck, Causmaecker, & Berghe, 2013) by focusing on the single heuristic sets involved and on the distinct experimental limits. Other recent research in selection hyper-heuristics was introduced by Kalender, Kheiri, Özcan, and Burke (2013) in which a simulated annealing-based move acceptance method is combined with a learning heuristic selection algorithm to manage the single heuristics.

HyFlex and related systems use a selection hyper-heuristic approach which operates on complete candidate solutions, perturbing them to try to improve their quality. As such, solving an instance typically involves some search, although usually limited. The work presented in this paper instead uses a selection hyper-heuristic approach that constructs a solution incrementally, each step of which could be expressed as a simple lookup of what to do next. The approach uses significant search effort to create such an incremental solution-builder, but the amortized cost of generating solutions to unseen problems is then much lower than for HyFlex-type methods. This framework has also been used for solving Constraint Satisfaction Problems (Terashima-Marín, Ortiz-Bayliss, Ross, & Valenzuela-Rendón, 2008).

One of the possible limitations of this approach, as stated by Sim and Hart (2013), is that if the nature of the unseen problems changes over time, the system may need periodic re-training.

Other heuristic-selection mechanisms have been used, for example Cowling, Kendall, and Soubeiga (2000) used a choice function based on the performance of single and pairs of heuristics. Burke, Petrovic, and Qu (2006) employed a case-based reasoning approach to tackle timetabling problems, while Bai, Blazewicz, Burke, Kendall, and McCollum (2012) proposed a learning approach by updating the heuristic selection weights depending on the heuristic performance after each learning period. Walker, Ochoa, Gendreau, and Burke (2012) used HyFlex to tackle a large set of instances within the domain of Vehicle Routing Problem by using two adaptive variants of a multiple neighborhood iterated local search algorithm.

3. The bin packing problem

The cutting and packing problem has been studied since 1939 (Kantorovich, 1960), even though a more intensive research started after the middle of the twentieth century. In 2007, Wäscher, Hausner, and Schumann (2007) suggested a complete problem typology which is an extension of Dychoff (1990). In that work, authors state that, in general terms, cutting and packing have a common identical structure given by a set of large objects that are to be filled and a set of items with which to do the filling, without overlapping other items or the edges of the objects.

In this paper, we consider the following problem types in Wäscher et al. typology: (a) the 1D single bin size bin packing problem, (b) the 2D regular single bin size bin packing problem as well as (c) the 2D irregular single bin size bin packing problem.

In the 1D BPP, there is an unlimited supply of bins, each with capacity $c > 0$, and a set of n items (each one of size $s_i < c$) is to be packed into the bins. The aim is to minimize the total number of bins used. In the 2D BPP, there is a set $L = (a_1, a_2, \ldots, a_n)$ of pieces to pack and an infinite set of identical rectangular *objects* into which the pieces are to be packed. The aim is to minimise the number of objects needed. A *problem instance* $I = (L, x_0, y_0)$ consists of a list of elements L and object dimensions x_0 and y_0. The term '2D regular BPP' is mainly used when all pieces are rectangular and the term '2D irregular BPP' refers to the case where pieces can be polygonal, not just rectangular. We deal only with the off-line BPP, in which the list of pieces to be packed is given in advance.

There are very few studies available of the *2D irregular BPP*, despite its practical importance. Ponce-Pérez, Pérez-García, and Ayala-Ramírez (2005) proposed a genetic algorithm based approach and tested it on one four-piece instance. Okano (2002) solved three real instances from a problem more general than our 2D irregular single bin size BPP, using variable bin sizes, where the solution involves finding appropriate sizes of material objects (bins) among given standard sizes in order to reduce waste. Babu and Babu (2001) presented the solution for one instance in which the objects are all of different size and shapes.

For all non-trivial BPP problems, exhaustive search is impractical and heuristic methods are needed. For the 2D BPP, heuristic methods typically involve iterating two actions: first, selecting both the next piece to be placed and the corresponding object in which to place it; and then, placing the selected piece in a position inside the object according to a given criterion. Some approaches may choose to include local search in these steps. For the 1D BPP, the placement step is unnecessary. In our approach, the two actions are done while *working with partial solutions* because heuristics construct a layout piece by piece, and feasibility is embedded into the heuristic since each piece is placed in a feasible position on the stock sheet and not moved thereafter.

Some metaheuristics for the 1D BPP have been implemented (Ducatelle & Levine, 2001; Falkenauer, 1996; Rohlfshagen & Bullinaria, 2010). For the 2D cutting and packing problem, Hopper and Turton (2001) reviewed metaheuristic implementations. Hyper-heuristic search has been applied to the 1D BPP (Pillay, 2012; Ross et al., 2003; Burke, Woodward, Hyde, & Kendall, 2007; Marín-Blázquez & Schulenburg, 2006; Sim, Hart, & Paechter, 2012) and different 2D BPPs (Terashima-Marín et al., 2006; Terashima-Marín et al., 2010; Burke, Hyde, Kendall, & Woodward, 2008; Garrido & Riff, 2007).

4. The evolutionary hyper-heuristic framework

The method presented in this paper is based on a genetic algorithm (GA) that evolves sets of condition-action rules that specify what to do next in any given problem state. Each individual in the GA population is a possible hyper-heuristic, specifying a complete set of such condition-action rules. This kind of GA-based approach has produced encouraging results in previous work (Ross et al., 2003; Terashima-Marín et al., 2006; Terashima-Marín et al., 2010).

According to the classification of hyper-heuristic approaches suggested by Burke et al. (2010a), our GA-generated hyper-heuristics fall into the category of selection hyper-heuristics because they select the single heuristics to be applied rather than generating new heuristics from components such as mutation and selection operators.

The key idea in our constructive approach is to build a complete solution by deciding what to do at each stage, including the initial stage. Here, *what to do* means choosing heuristics to place pieces and thus extend the solution. Each *stage* is described by a simplified representation of the problem state. We hypothesize that if two states are very similar then we would want to do the same thing in either state.

The hyper-heuristic has the following general form. Any state of the partially-solved problem is described using a fixed, simplified numeric representational scheme that expresses the state as a point in a unit hypercube of some sort. The GA's task is to find a representative set of points within, or close to, this hypercube and to label each such point with a specific heuristic step. This set of labelled points represents the hyper-heuristic, and it is utilized as follows. Until a solution is completely constructed, repeatedly: (a) encode the current problem state as a vector P; (b) find the labelled point L that is closest to P (this could be handled by dividing the space into tiny cubes and simply doing a lookup); (c) apply the heuristic step specified by that label to extend the solution. The question of what simplified representation to use is addressed in Section 5.4.

The type of GA used is a so-called 'messy' GA (Deb & Goldberg, 1991; Goldberg, Deb, & Korb, 1989), because of its variable length chromosomes. Each chromosome is composed of a series of *blocks*. Each block represents one labelled point, and consists of a vector identifying a point in the hypercube, and a final integer that identifies a heuristic by indexing a fixed array of available heuristics. A chromosome therefore consists of a number of points in a simplified state space, each point being labelled with a single heuristic, exemplified in Fig. 1 which shows six labelled points $h_1 \dots h_6$ and three successive problem states P, P', P'' in an overly simplistic three-dimensional hypercube. Clearly, not every point in such a hypercube will represent a valid state. For example, it is not very likely that (say) the features *percentage of small items* and *average of items sizes* would have large values at the same time.

Available problem instances are divided into separate training and testing sets. In the GA, chromosomes are evaluated using only a few instances from the training set. The testing set is reserved for evaluating the performance of the end result of the evolutionary process.

4.1. The fitness function

Each hyper-heuristic (chromosome) and each individual heuristic can be evaluated on any given problem in the same way:

$$Q = \frac{\sum_{i=1}^{N} U_i^2}{N} \tag{1}$$

where N is the total number of objects used and U_i is the fractional space utilization for each object i, so, $0 \leqslant U_i \leqslant 1$. We seek to maximize Q. A simple application of Lagrange multipliers shows that Q is minimized when all the U_i are equal (empty space is shared out equally), and because Q is concave upwards, simple geometric considerations show that Q is maximized at some boundary point when as many U_i as possible are as large as possible. Now, let $Q_k(H)$ be the performance of algorithm H on problem k, as defined in (1); if the fixed heuristics are $F_1, F_2, \dots F_6$ then define B_k to be the F_i that maximizes $Q_k(F_i)$, i.e. the fixed heuristic that gives the best performance on problem k.

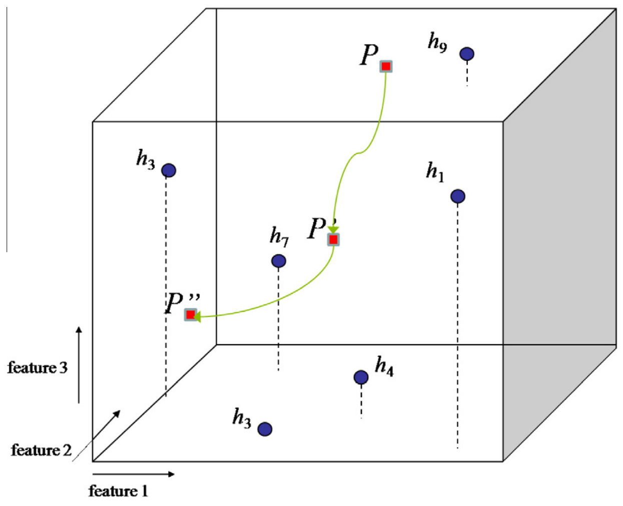


Fig. 1. Each block in a chromosome represents a point in the space of states labelled with a single heuristic. The hyper-heuristic solves a problem instance by finding the closest single heuristic at every solution stage.

Each chromosome is evaluated on some number m of problems and its fitness is computed as the average difference between the solution quality obtained by the chromosome with respect to the result given by the best single heuristic for every particular instance:

$$f(HH) = \frac{\sum_{k=1}^{m}(Q_k(HH) - Q_k(B_k))}{m} \quad (2)$$

Note that $f(HH)$ could be negative if the hyper-heuristic average performance is below the performance of the best heuristic for each instance solved. We expect that the evolutionary process finds a chromosome with positive value of $f(HH)$.

4.2. The GA cycle

The evolutionary process is given by the following steps:

1. Generate the initial population. Each individual is comprised of 10 to 15 blocks.
2. Randomly assign $m = 5$ problems from the training set to each chromosome and compute the chromosome fitness based on those problems (Eq. (2)).
3. Apply selection with tournament of size 2, crossover and mutation operators to produce two children.
4. Randomly assign $m = 5$ problems to each new child and estimate its fitness.
5. Replace the two worst individuals with the new offspring provided they are of better fitness.
6. Assign one additional and new problem to every individual in the new population and update fitness. Now the fitness on each individual is based on the number of instances it has seen so far; at age $T \geqslant 0$, a chromosome's evaluation is based on a total of $m = 5 + T$ problems.
7. Repeat from Step 3 until a termination criterion is reached.

If the GA is run with a small population of size P, this means any chromsome is unlikely to be assessed on very much more than about $5 + P/2$ problems. However, the chromosome's ancestors will have collectively seen very many more than this.

The GA uses two crossover and three mutation operators. These operators were taken from the previous implementation of the solution model for the 1D BPP (Ross et al., 2003) and the 2D regular BPP (Terashima-Marín et al., 2006; Terashima-Marín et al., 2010). The probability for applying each type of crossover or mutation operator was suggested by some early testing in the investigation for the 1D case (Ross et al., 2003).

Both of the following crossover operators employed have the same probability of being chosen. The one-point crossover operator works at block level, and 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, and vice versa. This operator shuffles blocks, so the blocks passed from a parent to an offspring are not in consecutive order; that is, gene linkage is broken. The two-point crossover operator is very similar to the normal two-point crossover. For each individual, we first select two blocks and then a point inside each block is chosen. Since the number of blocks in each chromosome is variable, the cut points in each parent are independently chosen. However, the points selected inside each corresponding block are forced to be the same for both parents, to avoid changing the meaning of any numbers, so that the recombination produces an exact number of blocks. The blocks and points are chosen using a uniform distribution. If one parent has a length of up to two blocks, then the individual is completely removed and recreated randomly with a number of blocks from 10 to 15. The other selected individual for crossover is copied exactly. The idea is to penalize chromosomes with a very small number of blocks.

Three mutation operators are used: add-block, remove-block, and in-block. Mutation is applied with probability 0.1, and the specific operator is randomly selected from along these three choices: in-block is twice as likely to be selected as the other two. The add-block mutation operator randomly generates a new block and adds it at the end of the chromosome, unless the length is 20 or more in which case a random block is deleted instead. The remove-block mutation operator randomly selects and eliminates a block within the chromosome unless the length is 6 blocks or less, in which case it adds a new one. The in-block operator randomly selects a position inside a random block and mutates it. If the selected position is a heuristic specifier, a random one is chosen. If it is a hypercube co-ordinate, a new value is chosen from an $N(0.5, 0.5)$ distribution and truncated to lie in the range -2 to 3.

Experiments in this investigation were carried out using a population size of 30, crossover probability of 1.0, mutation probability of 0.10, and 80 generations. These parameters worked well in preliminary experimentation and were used in previous studies (Terashima-Marín et al., 2010).

5. Implementation methodology

This section describes the implementation details including how the set of single heuristics and instances was gathered and the process for getting an adequate problem-state representation scheme.

5.1. Set of heuristics used

The following six heuristic approaches were employed. Heuristics are criteria to select the next piece to be placed and the corresponding object to place it. The 1D and 2D cases of the BPP share the same selection heuristics. For the 2D BPP, a placement heuristic was additionally used.

1. *First Fit Decreasing (FFD).* Considers the open objects in the order they were opened and places the largest piece in the first object where it fits. If the piece does not fit into any open object, FFD opens a new object to pack the piece.
2. *Filler.* Sorts the unplaced pieces in decreasing order of size and places as many pieces as possible within the open objects. If no single piece could be placed, it opens a new object and places as many pieces as possible in it.
3. *Best Fit Decreasing (BFD).* Sorts the unplaced pieces in decreasing order of size and places the next piece in the open object where it best fits, that is, in the object that leaves minimum waste. If the piece does not fit into any open object, BFD opens a new object to pack the piece.
4. *Djang and Finch with initial fullness of 1/4 ($DJD_{1/4}$).* Places items in an object, taking items by decreasing size, until the object is at least one-fourth full. Then, it initializes $w = 0$, a variable indicating the allowed waste, and looks for combinations of one, two, or three items producing a waste w or less. If any combination fails, it increases w by one twentieth of the object. When w reaches the amount of free space of the object, a new object is opened.
5. *Djang and Finch with initial fullness of 1/3 ($DJD_{1/3}$).* Same as $DJD_{1/4}$, once the object is full up to 1/3 it tries combinations of pieces.
6. *Djang and Finch with initial fullness of 1/2 ($DJD_{1/2}$).* Same as $DJD_{1/4}$, once the object is full up to 1/2 it tries combinations of pieces.

All of these are one-step constructive heuristics for the offline BPP. These approaches can often produce reasonable quality solutions with little computational cost (Bennell & Oliveira, 2009). $DJD_{1/4}$, $DJD_{1/3}$ and $DJD_{1/2}$ are variations of the DJD heuristic (López-Camacho, Ochoa, Terashima-Marín, & Burke, 2013). In preliminary experimentations, it has been found that $DJD_{1/4}$, $DJD_{1/3}$ and $DJD_{1/2}$, though similar, present a different behavior in different types of problem instances. For example, 2D instances with huge pieces are generally better solved by $DJD_{1/4}$; while $DJD_{1/3}$ is better with instances with small pieces (averaging below 1/10 of the object area). These heuristics were selected from a larger set, because they produced the best single-heuristic results in a preliminary study.

For 2D problems, the heuristic called **Constructive Approach with Maximum Adjacency (CAD)** was employed for finding the actual placement of the selected piece in a valid position inside the object. This heuristic is partially based on the approach suggested by Uday, Goodman, and Debnath (2001) and adapted by Terashima-Marín et al. (2010). It explores several possible positions and the one with the largest adjacency i.e. the common boundary between its perimeter and the placed pieces and the object edges is selected as the position of the new piece. In case of tie, the most bottom-left position is chosen. This heuristic was chosen because of its good performance in López-Camacho et al. (2013). There are several approaches to handle the geometry of irregular shapes, such as the nofit polygon or the phi function (Bennell & Oliveira, 2009; Alvarez-Valdes, Martinez, & Tamarit, 2013; Bennell, Scheithauer, Stoyan, & Romanova, 2010). In particular, our implementation is based on trigonometry (López-Camacho et al., 2013).

Previous studies had included in their heuristic repository every possible combination of several selection and placement heuristics considered, without performing any quality filter. Therefore, a very long list of heuristics comprised the heuristic repository (Terashima-Marín et al., 2006; Terashima-Marín et al., 2010). As a consequence, after the hyper-heuristics were built, most of the single heuristics were not called when solving a large set of instances. We conjecture that the presence of poor-quality heuristics delays the evolutionary process because it starts with an initial population with many unnecessary poor-quality hyper-heuristics and it takes time to weed them out.

5.2. Testbed instances

Our experimental testbed is comprised of a total of 1417 instances which are summarized in Table 1.

The 397 one-dimensional problem instances are from the literature. The first eight types of 1D instances in Table 1 are from Scholl, Klein, and Jürgens (1997), where we chose one out of every four instances in Scholl's data bases 1 and 2. Next instances come from Wäscher and Gau (1996) and the last four types of 1D instances are triplets from Falkenauer (1996) whose optimal solutions have exactly 3 items per bin with zero waste. These instances that are triplets originally had item sizes rounded to one decimal place. They were scaled up by a factor of 10 so that all values are now integers.

We have 540 two-dimensional instances with only convex polygonal pieces that were randomly generated by Terashima-Marín et al. (2010). This testbed also includes 30 rectangular instances (type *Conv I*). All the 2D convex instances, except type *Conv G*, have an optimum with zero waste; therefore, in the optimum solution, all objects must be filled up to 100%.

The 480 new 2D instances containing some non-convex polygons were randomly produced for this study. Of these, 240 were generated splitting at least five pieces from each instance from types *Conv A*, *Conv B*, *Conv C*, *Conv F*, *Conv H*, *Conv L*, *Conv M* and *Conv O*, respectively. Convex pieces from these instances were randomly selected and split into two pieces: one convex and one non-convex. The other 240 of the non-convex instances were produced by creating new convex instances and then splitting some of the pieces into non-convex polygons. Types *NConv U*, *NConv W* and *NConv X* were produced by splitting some pieces from instances that already had non-convex pieces. The 2D irregular instance set used in this study can be found at http://paginas.fe.up.pt/~esicup/tiki-index.php.

There is a variety of instance feature values in our experimental testbed. For example, there are instances whose pieces have an average size of 1/30 of the object, while other instances have huge pieces (averaging almost 2/3 of object size). Either the optimum number of objects or the best-known result ranges between 2 and 273. In the 2D problems, rectangularity (the ratio of the area of a piece to the area of the smallest enclosing rectangle) varies between 0.35 and 1.

Table 1
Description of problem instances.

1D			Convex 2D			Non Convex 2D		
Type	Num. of instances	Num. of pieces	Type	Num. of instances	Num. of pieces	Type	Num. of instances	Num. of pieces
DB1 n1	45	50	Conv A	30	30	NConv A	30	35–50
DB1 n2	45	100	Conv B	30	30	NConv B	30	40–52
DB1 n3	45	200	Conv C	30	36	NConv C	30	42–60
DB1 n4	45	500	Conv D	30	60	NConv F	30	35–45
DB2 n1	30	50	Conv E	30	60	NConv H	30	42–60
DB2 n2	30	100	Conv F	30	30	NConv L	30	35–45
DB2 n3	30	200	Conv G	30	36	NConv M	30	45–58
DB2 n4	30	500	Conv H	30	36	NConv O	30	33–43
Wäscher	17	57–239	Conv I	30	60	NConv S	30	17–20
Trip60	20	60	Conv J	30	60	NConv T	30	30–40
Trip120	20	120	Conv K	30	54	NConv U	30	20–33
Trip249	20	249	Conv L	30	30	NConv V	30	15–18
Trip501	20	501	Conv M	30	40	NConv W	30	24–28
			Conv N	30	60	NConv X	30	25–39
			Conv O	30	28	NConv Y	30	40–50
			Conv P	30	56	NConv Z	30	60
			Conv Q	30	60			
			Conv R	30	54			
Total	397		Total	540		Total	480	

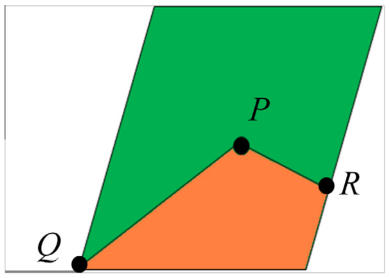


Fig. 2. One convex and one non-convex polygon created by the developed algorithm.

Table 2
Cluster membership for the 1D and 2D instances. According to fitness of the six single heuristics considered.

Cluster	1D	Convex 2D	Non convex 2D	Total
C1	224	209	287	720
C2	14	58	48	120
C3	4	29	36	69
C4	85	43	41	169
C5	69	103	81	253
C6	1		17	18
C7		19	15	34
C8		19	15	34
Total	397	480	540	1417

5.3. Algorithm for producing random instances with non-convex pieces

The process for creating problems with non-convex pieces starts with a set of convex pieces, each of which is defined by a set of corners that have integer co-ordinates. Then, a selected convex piece can be split into two pieces, one convex and other concave, by (a) selecting two edges; (b) for each of these two edges, either choosing an integer-valued interior point or (if none exist) choosing one of the endpoints of the edge, thus obtaining two points Q and R on the border of the piece; (c) choosing an integer-valued interior point P, and joining Q to P to R and then splitting the piece into two, one of them being concave (see Fig. 2).

Finally, the algorithm randomizes the order of all the pieces so that the two parts of a split piece are very unlikely to be adjacent in the list of pieces.

When this algorithm is applied to an instance that already has non-convex shapes it is possible to produce U-shaped polygons or even shapes with an internal empty space reached by a smaller entrance (resembling letter G). Shapes with holes are not produced by the algorithm.

5.4. Developing a problem-state representation for the testbed instances

We need to summarize each instance state by a numerical vector that quantifies some of its relevant features. Thus, the aim is to find a feature set that correlates fairly well with performance of the six heuristics. A great deal of the performance of a selection hyper-heuristic model may depend on the choice of representation of the problem state and the choice of the particular set of heuristics used (Ross, 2014). Messelis and Causmaecker (2014) have also recently argued that it is crucial to find a set of instance features that captures the internal structure of a problem domain and that is related to the performance of an algorithm. Finding such a set is not obvious. We have adopted the data mining based methodology proposed by López-Camacho, Terashima-Marín, and Ross (2010), in order to select the most representative features to characterize an instance of any type (1D, 2D regular, and 2D irregular).

The general methodology comprises six steps that we applied as follows:

Step 1. Each instance is solved by each of the six single heuristics and its performance is computed with Eq. (1).

Step 2. The performance of the six single heuristics for each instance is considered as a vector in $\mathbb{R}^6$, normalized to have length 1. Normalization removes the distinction between *easy* or *hard* instances in the measure of performance so that only the relative performance of the various heuristics matters.

Step 3. All instances are grouped into homogeneous clusters based on the normalized performance of the six single heuristics. The clusters will be used in Step 6, to find those problem-state features that are significant in predicting the clusters. We chose the k-means clustering technique which is a widely used algorithm (Arthur & Vassilvitskii, 2006). In this procedure the number of clusters has to be provided by the user, however there exist some approaches for selecting a good number of clusters. In our research, the number of clusters was eight and was determined according to the Hartigan criteria (Chiang & Mirkin, 2007). With the number of clusters chosen, 30 random k-means initializations were run, and we decided to use the seed that produces clusters which minimize the total intra-cluster variance (squared error function), given by: $\sum_{i=1}^{k}\sum_{x_j\in S_i}(x_j-\mu_j)^2$ where there are k clusters S_i, $i=1,2,\ldots,k$, and μ_i is the centroid or mean point of all the points $x_j \in S_i$. After obtaining the number of instances associated to each cluster, Table 2 summarizes the cluster membership according to dimensionality and convexity. Note that many instance types are split into different clusters. Instances in the same cluster have similar behavior when solved by the six heuristics considered and we attempt to find which features these instances have in common.

Step 4. This step consists of finding instance features that may be relevant to the ability of the heuristics to solve each instance. 23 numerical features were computed for each instance. The last two features are devoted to measure concaveness. The features are: (1) Number of pieces, (2) Mean number of sides for the instance pieces, (3) Variance of the number of sides of all instance pieces, (4) Mean area for the instance pieces (area for each piece is measured as a fraction of the object total area), (5) Variance of the area of all instance pieces, (6) Mean height for the instance pieces (height for each piece is measured as a fraction of the object height with the difference between its maximum and minimum y coordinates), (7) Variance of the height of all instance pieces, (8) Mean width for the instance pieces (width for each piece is measured as a fraction of the object width with the difference between its maximum and minimum x coordinates), (9) Variance of the width of all instance pieces, (10) Mean rectangularity for the instance pieces, (11) Variance of the rectangularity of all instance pieces, (12) Mean ratio (largest side)/(smallest side) for the instance pieces, (13) Variance of the ratio (largest side)/(smallest side) of all instance pieces, (14) Percentage of large pieces (whose area is greater than 1/2 of the object total area), (15) Percentage of small pieces (whose area is less than or equal to 1/4 of the object total area), (16) Percentage of right internal angles (respect to the total angles of all pieces of the instance), (17) Percentage of vertical/horizontal sides (respect to the total sides of all pieces of the instance), (18) Percentage of high rectangularity pieces (items which rectangularity is greater than 0.9), (19) Percentage of low rectangularity pieces (items which rectangularity is less than or equal to 0.5), (20) Percentage of non-convex pieces, (21) Average of the largest internal angle of all instance pieces, (22) Mean of the degree of concavity of the instance pieces (explained below), (23) Average of the proportion (area of piece)/(area of convex hull) for all instance pieces (explained below).

All items in 1D instances have only one dimension (height), so, their width has a variance of zero, meanwhile 2D instances will have a width variance greater than zero. For 1D instances, the area is computed assuming all items and bins have a fixed width, which means that area is proportional to height. All 1D items and 2D rectangles have rectangularity of 1.

The degree of concavity is defined as the concaveness of the **largest** internal angle and it can be computed by $DC = \frac{B}{A}$ (see Fig. 3) (Wang, 1998). For 1D items and 2D convex polygons (including rectangles), the degree of concavity is equal to 1. The degree of concavity for a concave polygon is more than one.

The convex hull of a given set *S* of points in the plane, is the smallest convex polygon that contains all of the points of *S*. The area of the convex hull for a non-convex polygon is greater than the area of the polygon, so the relation (area of piece)/(area of convex hull) is less than one only when dealing with non-convex polygons.

Step 5. Eliminate features that are highly correlated. It may happen that some couple or small sets of features are highly correlated (positive or negative) since they carry almost the same information. In this case, we can say we have equivalent features. The Pearson correlation coefficient (Rodgers & Nicewander, 1988) measures linear dependence between a pair of variables and it is an immediate way to perform this (although other measures of association exist (Joe, 1989)). For example, in the 2D irregular BPP *average area of pieces* and *percentage of small pieces* may be strong and negatively correlated. This can happen for pairs or even for small sets of features where all features in the set are highly correlated with all the others. For every pair or set of equivalent features it is possible to choose just one of them which would *act* as the representative of the other(s) and *delete* the others reducing the total set of features from the 23-feature list generated in Step 4, 17 features were kept.

Step 6. Select the final set of features among the set of the 17 problem features using Multinomial Logistic Regression (MLR) (Glonek & McCullagh, 1995).

Upon applying the complete methodology, nine significant features comprise the numerical representation. We added a tenth feature regarding the fraction of the instance total items that remain to be packed, so the GA learning process can have a sense of how advanced the solution of a given instance is. We performed a linear mapping to a fixed scale making each possible feature value to fall inside the range from 0 to 1, so each numerical term has the same weight. This numerical representation is capable of discriminating among the different categories of instances.

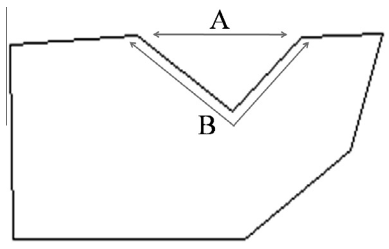


Fig. 3. Degree of concavity.

6. Experimental design and results

This section presents the experimental setup as well as the main results obtained along with the corresponding discussions and concluding remarks.

A series of experiments were designed in the present study by arranging the 1417 available instances into two balanced training and testing sets. In the Experiment 1, the training set was formed by the following instance types presented in Table 1: *DB1 n1* through *DB1 n4*, *Wäscher*, *Conv A* through *Conv I* and types *NConv A* through *NConv O*. In Experiment 3, every second instance from the testbed was selected to form the training set. Therefore, half of instances of every available type are included in the training set making training and testing sets very similar. Experiments 2 and 4 swap the training and testing sets from Experiments 1 and 3, respectively. Overall, four experiments were conducted.

For each of the four experiments, five GA processes were run using different seed values for making the random choices involved. Each run outputs two single hyper-heuristics (the best performers), and each evaluated on the unseen testing set. This is done because the fitness of every individual of a GA run is just an estimate since it is computed after solving a sample of problem instances (from the training set) during the evolution process. The hyper-heuristic with the best performance is chosen as the hyper-heuristic of the run. Overall, 20 hyper-heuristics were employed to measure the effectiveness of the model, that is, five for each experiment.

6.1. Comparing results against the best single heuristics

Table 4 shows a hyper-heuristic generated by the first run of Experiment 1. It has eight blocks and may employ up to five different single heuristics (actions) when solving a given problem instance. Features numbered from 1 to 10 are described in Table 3. In this example, when the hyper-heuristic is used to solve the Experiment 1 testing set, it only employs two different actions: heuristics 1 and 3 (Filler and $DJD_{1/4}$). This is because the other blocks represent problem states that were not reached by the instances solved. Most of the testing instances were solved using a combination of heuristics 1 and 3, and only 39 solutions of 2D instances were constructed using one single heuristic.

In general, the hyper-heuristics generated have an average of 11.2 blocks. Since our heuristic repository has 6 single heuristics, some of them may appear several times in a hyper-heuristic. The

Table 3
Representation of the instance state.

Feature	Description
1	Number of pieces
2	Mean area of remaining pieces
3	Variance of the area of remaining instance pieces
4	Mean of the rectangularity of remaining pieces
5	Variance of the rectangularity of remaining pieces
6	Mean of the height of the remaining pieces
7	Variance of the width of the remaining pieces
8	Fraction of remaining pieces in the instance whose area is above 1/2 of the object area
9	Mean of the degree of concavity of the remaining pieces
10	Fraction of the instance total items remaining

Table 4
Hyper-heuristic generated in the first run of Experiment 1.

Block	Feature										Action
	1	2	3	4	5	6	7	8	9	10	
1	1.03	1.33	0.80	0.49	0.64	0.22	0.81	0.43	0.91	0.65	0 (FFD)
2	−0.08	0.16	0.77	0.17	0.43	0.74	−0.34	0.37	0.92	0.30	1 (Filler)
3	1.06	0.89	0.40	−0.36	0.70	0.51	−0.03	0.84	1.39	0.36	2 (BFD)
4	0.34	0.91	0.56	0.27	0.41	0.93	0.87	0.82	0.91	0	2 (BFD)
5	1.08	1.25	0.83	1.26	0.51	0.04	0.49	0.02	0.70	0.91	0 (FFD)
6	0.57	0.10	0.46	0.52	0.67	0.39	0.87	0.44	−0.61	0.43	3 ($DJD_{1/4}$)
7	0.52	0.80	1.14	0.34	0.52	0.33	0.80	1.05	0.17	−0.42	4 ($DJD_{1/3}$)
8	0.70	1.08	0.87	−0.23	0.52	−0.59	0.89	0.23	0.55	0.31	2 (BFD)

six single heuristics considered were employed by at least one of the hyper-heuristics generated along the experiments.

Each one of the twenty generated hyper-heuristics employed a combination of two to five single heuristics when solving the testing set of the corresponding experiment. Complete results are shown in Table 5. Figures in cells indicate the percentage of instances that employs a particular number of extra objects (left column) when compared against results provided by the best single heuristic for each instance. We present results averaging the five best hyper-heuristics, each selected from each complete run in the experiment, as well as the results on the average of the two-best hyper-heuristics and the best hyper-heuristic per experiment, considering the testing instance set. In general, the average performance of the five hyper-heuristics produced per experiment is favorable since it clearly beats four of the single heuristics and it is very competitive against the other two, $DJD_{1/4}$ and $DJD_{1/3}$. Now, if we observe the performance of the two-best hyper-heuristics (average) and the best hyper-heuristics per experiment, the results definitively are better than those provided by any single heuristics. Two important aspects we can point out here: there is a saving in the number of objects for a percentage of the instances, and the largest number of extra objects used by them is no greater than

Table 5
Number of extra objects obtained by hyper-heuristics and single heuristics when compared against the results of the best single heuristic for each instance (figures indicate percentage of cases). Zero values are displayed as blank cells.

Extra objects	Hyper-heuristics			Single heuristics					
	Average	2B-Avg	Best	FFD	Filler	BFD	$DJD_{1/4}$	$DJD_{1/3}$	$DJD_{1/2}$
Experiment 1									
⩽ −2		0.6	0.7						
−1	0.6	2.8	2.7						
0	86.2	93.4	94.9	62.1	63.0	64.5	96.1	93.0	72.0
1	13.2	3.2	1.7	23.0	22.0	20.7	3.9	6.6	14.8
2				3.9	4.1	3.8		0.4	2.8
3				1.1	1.1	1.1			0.8
⩾ 4				9.9	9.9	9.9			9.6
Experiment 2									
⩽ −2									
−1									
0	92.6	98.2	100	70.2	69.4	70.6	96.3	95.8	72.8
1	7.4	1.8		21.1	21.8	20.5	3.5	4.2	19.5
2				6.2	6.2	6.4	0.1		6.1
3				2.4	2.4	2.4			1.6
⩾ 4				0.1	0.1	0.1			
Experiment 3									
⩽ −2			0.3						
−1		1.1	0.8						
0	94.6	95.9	96.9	66.1	66.1	67.2	96.0	94.1	72.2
1	5.4	3.0	2.0	21.6	21.6	20.3	3.8	5.6	17.1
2				5.4	5.4	5.5	0.1	0.3	4.8
3				2.0	2.0	2.0			1.3
⩾ 4				4.9	4.9	4.9			4.7
⩽ −2			0.1						
−1		1.0	0.8						
0	91.7	95.6	97.9	66.1	66.3	67.8	96.3	94.6	72.6
1	5.1	3.4	1.1	22.4	22.1	20.9	3.7	5.2	17.2
2	1.6			4.8	4.9	4.7		0.1	4.1
3	0.6			1.6	1.6	1.6			1.1
⩾ 4	1.1			5.1	5.1	5.1			4.9
All experiments									
⩽ −2		0.1	0.4						
−1	0.1	1.2	1.6						
0	91.3	95.8	97.4	66.1	66.2	67.5	96.2	94.4	72.4
1	7.8	2.9	0.6	22	21.9	20.6	3.7	5.4	17.1
2	0.4			5.1	5.2	5.1	0.1	0.2	4.4
3	0.1			1.8	1.8	1.8			1.2
⩾ 4	0.3			5	5	5			4.8

one, usually for a small percentage of the instance set. For example, in Experiment 1 the average of the two-best and the best hyper-heuristic required one object less in 2.8% and 2.7% of instances, respectively, and two or more objects less in 0.6% and 0.7% of instances, respectively. For Experiment 2, the best hyper-heuristic obtains the same results that the best single heuristic 100% of times; which means that the hyper-heuristic learns to behave as the best single heuristic per problem. Along all experiments, the best hyper-heuristic delivered fewer objects than the best single heuristic for 2% of the instances. We could state that the hyper-heuristics produced are capable of learning to perform as well as the best single heuristic for each instance, and for some cases, even better. In other words, these hyper-heuristics may be considered as general solving methods.

It has been confirmed that solving an instance with a hyper-heuristic is faster than solving it with each of the single heuristics and then choosing the best result (Terashima-Marín et al., 2010). We ran the algorithms on a 1.66 GHz PC with 1.98 GB of RAM. Once the hyper-heuristic is generated, it solves each instance in 20 s on average.

For most cases, the best hyper-heuristic achieves the same number of objects than the best single heuristic (higher percentages in Table 5 are in the 0-object row). We ran the non-parametric Mann–Whitney statistical test for means comparison of extra objects between 1D and 2D cases. We want to know if the hyper-heuristic model performance is different for 1D and 2D instances. For Experiment 1, the extra number of objects delivered by the best hyper-heuristic is statistically different for 1D and for 2D instances (p-value $= 0.001$). For the rest of the experiments, the difference is not significant between 1D and 2D BPP. When we perform a comparison of means test between results for convex and non-convex 2D instances, we found that there is a significant difference only in Experiment 4 (p-value $= 0.016$). In conclusion, most of the experiments show that hyper-heuristics performance is not statistically different for the distinct categories of BPP considered in this study.

It is difficult to establish a fair comparison between our approach and other studies since, in order to prove generality, we used a large set of instances, and many of them, i.e. the non-convex 2Ds, have not been used in previous studies. An exhaustive comparison with other algorithms would fall out of the scope of this investigation. However, in order to provide an idea of the performance of our algorithm over specific instances, we present a comparison versus an algorithm for solving 1D BPPs and tested it with some of the same instances that we employed within our approach (Mumford, 2008). Mumford designed a hybrid (or memetic) evolutionary algorithm specifically focused on solving set partitioning problems. She called it Genetic Simulated Annealing framework (GSA). New genetic operators were devised and a simulated annealing cooling schedule was adopted to maintain a balance between quality and population diversity. As a part of the evolutionary process, some grouping and reordering heuristics are applied to pre-process the chromosomes prior to crossover. Mumford employed order based representation, a population size of 300 and the GA run for 250 generations in her experimentation. Four instances from the Scholl databases and four 20-instance sets from Falkenauer were solved by both Mumford's and our approach. The summary of the results is shown in Table 6. In the first two Scholl instances, all approaches obtained the same results with one extra from the optimum solution. For the N4W1B1R0 instance, all approaches found the optimum number of objects. For the last of the Scholl instances, its best simple heuristic required an additional object while the hyper-heuristics employed two more objects. For the two smaller Falkenauer data sets, our approach obtained the same results that GSA. Taking into account that both our simple heuristics and our hyper-heuristics solve every instance in a single one pass without regrouping, we consider our results to be very competitive. The GSA is a more complex approach where a feasible solution is found many times during the GA run.

Table 6
Comparison of our approach versus Mumford's GSA for Scholl and Faulkner instances.

Instance	Number of objects			
	Optimum	GSA	Best simple heuristic	Best hyper-heuristic
N4C3W2A	203	204	204	204
N4C3W4A	216	217	217	217
N4W1B1R0	167	167	167	167
N4W3B1R0	71	71	72	73
Trip60	20	21	21	21
Trip120	40	41	41	41
Trip249	83	84	84.8	84.75
Trip501	167	168	170.8	170.05

6.2. Frequency of usage of single heuristics per instance category

There is a correspondence between the category of problem instances and the single heuristics more often employed within a hyper-heuristic. This is what we expected since different categories of instances have different numerical representations; so, the hyper-heuristics suggest different single heuristics to apply. Fig. 4 illustrates this fact averaging all runs of Experiment 1. For 1D instances, the Filler heuristic was employed 29.1% of the times, while this heuristic was chosen only 7.5% of the times when solving 2D convex instances. We ran a test of contingency table with the χ^2 statistic to verify this, concluding that the usage of single heuristics is indeed related to instance category (p-value

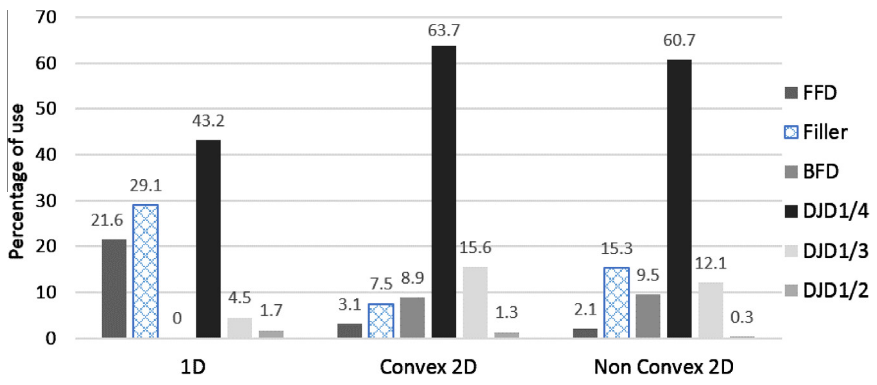
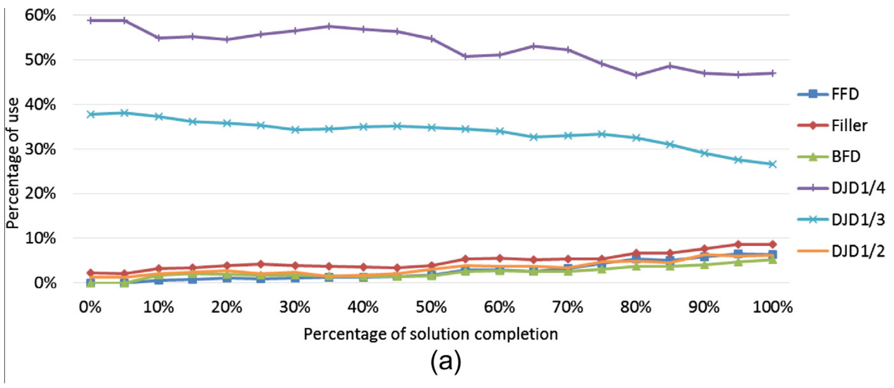


Fig. 4. Percentage of usage of the single heuristics when solving the testing set with hyper-heuristics of Experiment 1.



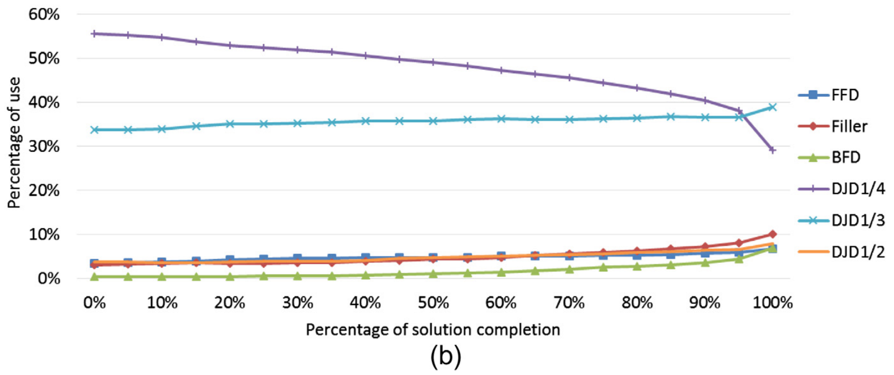


Fig. 5. Percentage of times that a heuristic is called during the solution construction process for (a) 1D and (b) 2D instances. All experiments averaged. Each instance is analyzed from piece 1 and up to its total number of pieces.

< 0.001). Also, there is a significant difference in the employment of single heuristics between the two types of 2D instances considered (convex and non-convex). We arrive at the same conclusion when considering the remaining three experiments (López-Camacho et al., 2013).

We have found hyper-heuristics that are able to solve well different kind of problem instances by automatically choosing the proper single heuristic for each type. It is worth mentioning that the heuristic $DJD_{1/4}$ was the most used in solving all the instances, which suggests that this is a very effective and robust heuristic. Additionally, the plots in Fig. 5 illustrate the percentage of times that each heuristic is called when solving the testing sets regarding the percentage on the completion of the solution. For this analysis we averaged all experiments. For both sets of instances, we observe that heuristics $DJD_{1/4}$ and $DJD_{1/3}$ are the most frequently used, but there is a decreasing tendency in their usage during the last stages of the solution construction process.

6.3. Comparing results for convex and non-convex instances

There are 240 convex instances in the testbed that have also their non-convex version. These are instances types *Conv A*, *Conv B*, *Conv C*, *Conv F*, *Conv H*, *Conv L*, *Conv M* and *Conv O* from Table 1. Their respective non-convex version have the same pieces, except for those that were split to generate concaveness (see Section 5.3). Optimal solutions of the non-convex instances have the same number of objects than their respective convex instances, with all objects filled up to 100%. For the 240 convex instances, the number of pieces goes from 28 to 40. The number of pieces chosen to be split in each of these convex instances goes from 5 to 24. We want to compare how single heuristics and hyper-heuristics solve problem instances that have several pieces in common. These results are summarized in Table 7. Figures in cells indicate percentage of instances where the best single heuristic and the best hyper-heuristic have employed fewer, equal or more objects to solve the non-convex version compared with the number of objects employed to solve each convex instance. For example, the best single heuristic per instance employed the same number of objects in 65% of the non-convex instances than the number of objects employed for their respective convex version. In 0.4% of the instances (which means only one case), the best single heuristic solved a non-convex instance employing fewer objects than its convex version. In the rest of the cases, approximately one-third, solving the non-convex instance requires more objects than solving the convex instance. Either instances are solved by the best single heuristic or by the best hyper-heuristic of any experiment, results are basically the same: in about one-third of the cases, solving a problem instance with split pieces requires more objects than solving the original instance. This may be due to the fact that our general-purpose methodology is not intended to match a piece with the concavity of other piece where it fits. We are dealing with a combination of fast single-pass constructive heuristics which means that all pieces have only one opportunity to couple with the pieces that perfectly complements them.

6.4. Alternation and interaction of single heuristics within hyper-heuristics

When a hyper-heuristic solves a given problem instance, it re-computes the problem state after every heuristic application. Every time that heuristics FFD, Filler or BFD are applied, only one piece is placed. In contrast, heuristics $DJD_{1/4}$, $DJD_{1/3}$ and $DJD_{1/2}$ place 1, 2 or 3 pieces (see Section 5.1). Most of the times, successive re-computations of problem states and application of the corresponding heuristic, results in an almost-unchanged problem state leading to the same block in the chromosome. Therefore, it is likely to apply the same heuristic in a sequence of steps. Moreover, several blocks may have associated the same heuristic, as it happens in the

Table 7
Solving non-convex instances compared against solving their convex version. Percentage of cases when non-convex instances require fewer, equal and more objects than convex instances.

	Best single heuristic	Best hyper-heuristic			
		Exp. 1	Exp. 2	Exp. 3	Exp. 4
Fewer objects	0.4	1.3	0.8	1.3	0.4
Same number of objects	65	64	63	64	63
More objects	34	35	37	35	37

Table 8
Percentage of single heuristic changes when solving all testing sets.

Heuristic changes	Number of pieces				All instances
	Up to 50	51–100	101–200	201–500	
0	46.9	48.7	34.6	33.4	45.4
1	27.6	28.0	38.7	37.2	29.3
2	10.0	7.7	8.9	9.7	9.2
3	8.4	7.4	7.6	6.6	7.9
4	2.8	3.2	3.1	4.6	3.1
$\geqslant 5$	4.4	4.9	7.1	8.5	5.1

hyper-heuristic shown in Table 4. Hence, it is also possible to select the same single heuristic even when changing blocks in the hyper-heuristic solution process. Averaging all our experiments, 46.9% of instances with up to 50 pieces were solved using one single heuristic from start to end (although the choice of this single heuristic varies from instance to instance); and, 27.6% of these instances have only one change of single heuristic when building the solution. This means that one heuristic was employed for placing the first pieces and then, another heuristic was chosen to finish placing the rest of the pieces. 10.0% of instances with 50 pieces or less involved 2 heuristic changes when solved by a hyper-heuristic. By contrast, there are few instances that were solved with up to 20 heuristic changes. Table 8 shows the results of the analysis of heuristic alternation. Note that several heuristic changes may imply that the hyper-heuristic is returning to single heuristics previously employed in the same problem instance.

We are interested in exploring whether the quality of solutions is related with the number of heuristic changes performed during the solution process. Table 9 summarizes results for all experiments to show this fact. Hyper-heuristics perform an average of 2.7 heuristic changes when solving instances that get one object less than the best single heuristic. The same hyper-heuristics make 1.1 heuristic changes when solving instances whose solution is the same that the best single heuristic. For those cases where hyper-heuristics solutions get more objects, more heuristic changes are done. We conclude that hyper-heuristics perform more heuristic changes for finding the best as well as the worst solutions. Hence, a hyper-heuristic that makes few heuristic changes will get a solution similar to one of the single heuristics. Hyper-heuristics find different solutions when they *dare* to combine a greater number of single heuristics. In general, with more changes between single heuristics, a better solution may emerge (with the risk of getting a worse solution, though).

Table 10 shows how long are the sequences of the same single heuristic before changing to another heuristic. For example, in instances with up to 50 pieces, these sequences have an average length of 16.9. This means that the same heuristic is applied an average of 16.9 times before the hyper-heuristic changes to another single heuristic. For further research, we suggest to use the same single heuristic a given number of times before recomputing the problem state. With this approach, we would expect to both reduce the computation time and keep the good results.

Also, we analyzed which sequences of single heuristics were performed during our experiments. We want to know which heuristics tend to follow others during the solution process. For example, for 1D instances, 23.1% of all heuristic changes were from heuristic $DJD_{1/3}$ followed by $DJD_{1/4}$ (see Table 11). $DJD_{1/4}$ is the first heuristic in 40.8% of all heuristic changes. Tables 12 and 13 show the corresponding results for 2D convex and non-convex instances respectively.

Notice that Tables 11–13 are highly asymmetric matrices. For instances from all types, heuristic BFD almost exclusively goes before and after heuristics $DJD_{1/4}$ and $DJD_{1/3}$. This means that heuristic BFD almost never pairs with heuristics FFD, Filler and $DJD_{1/2}$. Moreover, heuristics FFD, Filler and BFD never follow each other when solving 2D instances. These three heuristics place a piece one at a time, while the remaining heuristics ($DJD_{1/4}$, $DJD_{1/3}$ and $DJD_{1/2}$) place groups of 1, 2 or 3 pieces. In conclusion, place-one heuristics always alternate with the DJD heuristics.

With respect to the computational time, we can confirm that solving an instance with the hyper-heuristic is faster than solving the instance with each single-heuristic and then selecting the best result. Once the hyper-heuristic is generated, it solves each instance in 21 s in average. However, we are aware that the evolving process to generate the hyper-heuristic is much slower, since it is population-based and requires to solve many instances during the training phase. Table 14 shows a time comparison between the set of single-heuristics and the hyper-heuristics. The six single heuristics considered in this research solve instances with a huge variety of time length. For instance, the fastest single heuristic, FFD, solves 1D instances in 0.2 s per case, in average; while $DJD_{1/4}$ is the most time-consuming heuristic averaging 24.8 s per

Table 9
Average of heuristic changes.

Extra objects against best single heuristic	Number of pieces				All instances
	Up to 50	51–100	101–200	201–500	
$\leqslant -2$				2.3	2.3
-1	2.4	3.3	4.0	2.6	2.7
0	1.1	1.0	1.2	1.2	1.1
1	1.6	1.7	2	2.4	1.8
2	5.3	1.5	3.6	4.1	3.2
3	1.5	5.8	4.9	3.5	4.5
$\geqslant 4$		4.0	5.3	8.2	8.2

Table 10
Average length of single heuristic runs.

Number of pieces	Average length of heuristics runs
Up to 50	16.9
51–100	31.8
101–200	83.6
201–500	205.3
All instances	41.8

Table 11
Percentage of sequences of single heuristic pairs when solving 1D instances in all testing sets.

From heuristic	To heuristic						Total
	FFD	Filler	BFD	$DJD_{1/4}$	$DJD_{1/3}$	$DJD_{1/2}$	
FFD		0.1		0.8	2.0		2.9
Filler					8.6	0.7	9.3
BFD	0.6			1.8	0.6		3.0
$DJD_{1/4}$	1.5	6.7	5.7		10.7	16.2	40.8
$DJD_{1/3}$	4.4	6.9	1.1	23.1		0.4	35.9
$DJD_{1/2}$		0.3		7.5	0.4		8.2
Total	6.5	14.0	6.8	33.2	22.3	17.3	100

Table 12
Percentage of sequences of single heuristic pairs when solving 2D convex instances in all testing sets.

From heuristic	To heuristic						Total
	FFD	Filler	BFD	$DJD_{1/4}$	$DJD_{1/3}$	$DJD_{1/2}$	
FFD				2.6	0.2	3.8	6.6
Filler				4.7	4.6		9.3
BFD				0.8	2.7		3.5
$DJD_{1/4}$	7.1	12.6	2.9		8.0	19.0	49.6
$DJD_{1/3}$	0.6	4.3	6.7	10.5		0.5	22.6
$DJD_{1/2}$	2.2			5.6	0.5		8.3
Total	9.9	16.9	9.6	24.2	16.0	23.3	100

1D instance. The best single heuristic may be different for each particular case as we have mentioned in previous sections. Moreover, for many cases the smallest number of objects is obtained by several of the six heuristics. We averaged the recorded time employed by all single heuristics that provided the smallest number of objects per instance. Table 14 shows that best single heuristics employ more time than the average heuristic. For example, for 1D instances, the best of the single heuristics employed 21.1 s per case. The last two columns in the table regard hyper-heuristics. Each instance in the testbed was assigned to the testing set in two out of the four experiments. This means that each instance was solved 10 times by the hyper-heuristics (five hyper-heuristics were produced by each experiment, one for each complete run). The next to last column in Table 14 refers to the average of the time taken by the 10 hyper-heuristics that solve each instance, while the last column averages the times when the hyper-heuristic obtained the smallest number of objects. It is worth noting that

Table 13
Percentage of sequences of single heuristic pairs when solving 2D non-convex instances in all testing sets.

From heuristic	To heuristic						Total
	FFD	Filler	BFD	$DJD_{1/4}$	$DJD_{1/3}$	$DJD_{1/2}$	
FFD				2.6	0.2	3.1	5.9
Filler				3.7	6.3	0.5	10.5
BFD				0.9	1.9		2.8
$DJD_{1/4}$	6.1	9.9	3.5		9.4	19.3	48.2
$DJD_{1/3}$	0.8	5.9	5.2	10.5		0.5	22.9
$DJD_{1/2}$	1.8	0.1	0.1	7.4	0.5		9.9
Total	8.7	15.9	8.8	25.1	18.3	23.4	100

time obtained by the average hyper-heuristic is very similar to the time obtained by the best hyper-heuristic per case, and both times are larger than those obtained by any of the single heuristics. This may be explained by the fact that hyper-heuristics compute the numerical state after each application of a single heuristic. Nevertheless, Table 5 shows that results from the produced hyper-heuristics are better than the average results from single heuristics.

7. Conclusions and future work

In the present paper we have introduced an evolutionary selection and constructive hyper-heuristic approach that combines single heuristics in such a way that is able to solve efficiently a wide range of 1D and 2D bin packing problem instances and with no additional parameter tuning. The 2D set contains pieces with different shapes such as rectangles, convex and non-convex polygons. Different from the perspective of Ochoa et al. (2012) in terms of generality which is more oriented to cross-domain heuristic search, our framework also provides a step towards the development of general solvers for optimization problems, but considering different types of instances within the same domain. The model automatically selects the best heuristic for a given instance state during the solution process and its results are comparable, and sometimes even better, to those of competent single-heuristics when these are applied in isolation.

Our study has blended various techniques to produce a unified framework for solving an important combinatorial problem such as the BPP, with many practical applications in industry. Among the distinctive characteristics of the proposed framework we could list the following: the application of a messy-type GA to define a variable-length chromosomes, each representing a hyper-heuristic; the use of a data mining methodology for determining the relevant feature set in the problem-state representation; the offline analysis to determine the appropriate set of good-quality single heuristics as the potential choices of the hyper-heuristics; a random 2D non-convex problem instance generator; the use of a large benchmark set to test the framework; an experimental set up with a series of statistical tests to prove the significance of the results; and, an empirical analysis on the application of single heuristics within the hyper-heuristic model aimed at understanding

Table 14
Average computational time (in seconds) per category of instances.

	FFD	Filler	BFD	$DJD_{1/4}$	$DJD_{1/3}$	$DJD_{1/2}$	Simple heuristics		Hyper-heuristics	
							Average	Best	Average	Best
1D	0.2	29.3	5.1	24.8	24.0	24.1	17.9	21.1	21.9	21.8
2D-cvx	2.5	14.2	2.5	18.5	18.5	18.4	12.4	12.8	20.7	20.7
2D-Ncvx	2.9	9.2	3.5	18.3	18.3	18.3	11.8	12.9	20.6	20.6
Total	2.0	16.7	3.6	20.2	20.0	20.0	13.7	15.1	21.0	21.0

interesting aspects such as their frequency, alternations, and interactions. We found that hyper-heuristics tend to choose different single heuristics for different kind of instances. This is a sign that the evolutionary process has found that distinct instances states are more suitable to be solved with different single heuristics. We found some patterns related with the alternation of single heuristics.

Each of the hyper-heuristics produced can be considered as other possible heuristic. If there is enough time for finding a solution, the general recommendation is to solve the instance at hand with every single heuristic plus one or several hyper-heuristics, then choose the best result. If time is a constraint and we need to choose only one heuristic to find a solution, a good decision is to employ a hyper-heuristic to do so. Although our constructive approach has the advantage of providing fast solutions, it is not intended for cases when more time and computational resources are available. In those cases, intelligent perturbating strategies, as in HyFlex, may be applied after the solution is built searching for an improvement. In our study, performance of the hyper-heuristics and single heuristics is measured only in terms of how well the pieces are packed into the objects, i.e. minimizing the waste space and consequently the number of used bins, but other metrics might be taken into account, for instance, the speed of heuristics. In a related study (Gomez & Terashima-Marín, 2012), the 2D BPP problem has been tackled as bi-objective trying to find the trade-off between the number of bins and time used to place the pieces.

The fact that hyper-heuristics achieve better results than the best of the single heuristics justify the existence of hyper-heuristics beyond any simple heuristic, since for some applications like BPPs, any reduction in material is extremely valuable. In general, statistical analysis show that hyper-heuristics performance is not different across the different BPP types considered. So, our hyper-heuristics achieve some level of generality that crosses several types of BPP. The proposed framework could be adopted as well when solving the BPP with some common constraints, for example, the requirement of guillotinable cuts, different rotation allowed for different shapes, or object stocks of different size and material quality. A challenging idea would be to include 3D instances within the framework with the related implications regarding the representation, the set of heuristics, and the evaluation function. This would be a further step towards generality of hyper-heuristics when solving other types of BPPs.

The present work opens up interesting research directions that are briefly described in the following lines. A topic for future work would be that instead of choosing the closest point to a given problem state and then apply the labelled heuristic, we may develop a learning mechanism to select the most appropriate one from various options depending, for example, on previous performance, heuristic speed, as applied in other selection hyper-heuristic approaches (Kalender et al., 2013; Misir et al., 2013; Gomez & Terashima-Marín, 2012). Derived for the analysis of the frequency and alternation of single heuristics, the research can be extended to explore the application of the same single heuristic several times before recomputing the instance state for choosing another hyper-heuristic block. This would reduce the computational time with the expectation of having competitive results. All features employed in the instance representation were related to the pieces to be placed. But, one or several features of the instance state representation could actually describe the state of the open objects. This is because the suitability of one heuristic for solving an instance state may depend not only on the remaining pieces, but also, on the state of the objects partially filled. For example, in a new instance to be solved, all objects are empty; in contrast with an instance with few pieces remaining where several objects may have some free areas with different sizes and shapes. As far as we know, none of the works which have dealt with numerical representation of instances states for bin packing have considered this issue. For example, one numerical term in the representation vector could refer to the number of open objects available to choose from, or to the open objects percent of free area. The combination of our constructive method with other perturbative approaches provided in the literature would be worth exploring, as well, along with the corresponding comparative studies.

Acknowledgments

This research was supported in part by Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM) under the Strategic Project PRY-075 and the Consejo Nacional de Ciencia y Tecnología (CONACYT) Project under Grant 99695.

References

Alvarez-Valdes, R., Martinez, A., & Tamarit, J. (2013). A branch & bound algorithm for cutting and packing irregularly-shaped pieces. *International Journal of Production Economics., 145*, 463–477.

Arthur, D., & Vassilvitskii, S. (2006). How slow is the k-means method? In *SCG '06: proceedings of the twenty-second annual symposium on computational geometry* (pp. 144–153). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1137856.1137880.

Babu, A. R., & Babu, N. R. (2001). A generic approach for nesting of 2-D parts in 2-D sheets using genetic and heuristic algorithms. *Computer-Aided Design, 33*, 879–891.

Bai, R., Blazewicz, J., Burke, E. K., Kendall, G., & McCollum, B. (2012). A simulated annealing hyper-heuristics methodology for flexible support. *4OR: A Quarterly Journal of Operations Research, 10*, 43–66.

Bennell, J. A., & Oliveira, J. F. (2009). A tutorial in irregular shape packing problems. *Journal of the Operational Research Society, 60*, S93–S105.

Bennell, J. A., Scheithauer, G., Stoyan, Y., & Romanova, T. (2010). Tools of mathematical modeling of arbitrary object packing problems. *Annals of Operations Research, 179*, 343–368.

Bhatia, A. K., Hazra, M., & Basu, S. K. (2009). Better-fit heuristic for one-dimensional bin-packing problem. In *IEEE international advance computing conference (IACC)* (pp. 193–196). IEEE.

Burke, E. K., Woodward, J., Hyde, M. R., & Kendall, G. (2007). Automatic heuristic generation with genetic programming: evolving a jack-of-alltrades or a master of one. In *Genetic and evolutionary computation conference, GECCO'07* (pp. 7–11).

Burke, E. K., Hyde, M. R., Kendall, G., & Woodward, J. (2008). A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics. Technical Report NOTTCS-TR-2008-2. School of Computer Science and Information Technology. University of Nottingham.

Burke, E. K., Curtois, T., Hyde, M. R., Kendall, G., Ochoa, G., Petrovic, S., Rodríguez, J. A. V., & Gendreau, M. (2010). Iterated local search vs. hyper-heuristics: towards general-purpose search algorithms. In *IEEE Congress on Evolutionary Computation* (pp. 1–8).

Burke, E., Gendreau, M., Hyde, M., Kendall & Ochoa, G. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of Operational Research Society*, 1–30.

Burke, E. K., Gendreau, M., Ochoa, G., & Walker, J. D. (2011). Adaptive iterated local search for cross-domain optimisation. In N. Krasnogor & P. L. Lanzi (Eds.), *GECCO* (pp. 1987–1994). ACM.

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

Burke, E. K., Hellier, R. S. R., Kendall, G., & Whitwell, G. (2006). A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. *Operations Research, 54*, 587–601.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. (2010a). In *A classification of hyper-heuristic approaches. International series in operations research & management science* (Vol. 146, pp. 449–468). US: Springer. http://dx.doi.org/10.1007/978-1-4419-1665-5_15.

Burke, E. K., Hyde, M. R., Kendall, G., & Woodward, J. (2012). Automating the packing heuristic design process with genetic programming. *Evolutionary Computation, 20*, 63–89.

Burke, E. K., Petrovic, S., & Qu, R. (2006). Case-based heuristic selection for timetabling problems. *Journal of Scheduling, 9*, 115–132.

Chiang, M., & Mirkin, B. (2007). Experiments for the number of clusters in k-means. In J. Neves, M. F. Santos, & J. M. Machado (Eds.), *Progress in artificial intelligence. Lecture notes in computer science* (Vol. 4874, pp. 395–405). Berlin, Heidelberg: Springer. http://dx.doi.org/10.1007/978-3-540-77002-2_33.

Cowling, P. I., Kendall, G., & Soubeiga, E. (2000). A hyperheuristic approach to scheduling a sales summit. In E. K. Burke & W. Erben (Eds.), *PATAT. Lecture notes in computer science* (Vol. 2079, pp. 176–190). Springer.

Deb, K., & Goldberg, D. E. (1991). mGA in C: A messy genetic algorithm in C.

Ducatelle, F., & Levine, J. (2001). Ant colony optimisation for bin packing and cutting stock problems. In proceedings of the uk workshop on computational intelligence.

Dychoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research, 44*, 145–159.
Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics, 2*, 5–30.
Garrido, P., & Riff, M.-C. (2007). An evolutionary hyperheuristic to solve strip-packing problems. In *Proceedings of the 8th international conference on Intelligent data engineering and automated learning. IDEAL'07* (pp. 406–415). Berlin, Heidelberg: Springer-Verlag.
Glonek, G. F. V., & McCullagh, P. (1995). Multivariate logistic models. *Journal of the Royal Statistical Society. Series B (Methodological), 57*, 533–546.
Goldberg, D. E., Deb, K., & Korb, B. (1989). Messy genetic algorithms: motivation, analysis and first results. *Complex Systems, 3*, 493–530.
Gomez, J. C., & Terashima-Marín, H. (2012). Building general hyper-heuristics for multi-objective cutting stock problems. *Computación y Sistemas, 16*, 321–334.
Hopper, E., & Turton, B. C. H. (2001). A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artificial Intelligence Review, 16*, 257–300.
Hu-yao, L., & Yuan-jun, H. (2006). NFP-based nesting algorithm for irregular shapes. In *Symposium on applied computing* (pp. 963–967). New York, NY, USA: ACM Press.
Joe, H. (1989). Relative entropy measures of multivariate dependence. *Journal of the American Statistical Association, 84*, 157–164.
Kalender, M., Kheiri, A., Özcan, E., & Burke, E. K. (2013). A greedy gradient-simulated annealing selection hyper-heuristic. *Soft Computing, 17*, 2279–2292.
Kantorovich, L. V. (1960). Mathematical methods of organising and planning production. *Management Science, 6*, 366–422.
López-Camacho, E., Ochoa, G., Terashima-Marín, H., & Burke, E. K. (2013). An effective heuristic for the two-dimensional irregular bin packing problem. *Annals of Operations Research, 206*, 241–264.
López-Camacho, E., Terashima-Marín, H., & Ross, P. (2010). Defining a problem-state representation with data mining within a hyper-heuristic model which solves 2D irregular bin packing problems. In Á. F. Kuri-Morales & G. R. Simari (Eds.), *Advances in artificial intelligence IBERAMIA. Lecture notes in computer science* (Vol. 6433, pp. 204–213). Springer.
Marín-Blázquez, J. G., & Schulenburg, S. (2006). Multi-step environment learning classifier systems applied to hyper-heuristics. In *Conference on genetic and evolutionary computation. Lecture notes in computer science* (pp. 1521–1528). New York, NY, USA: ACM.
Messelis, T., & Causmaecker, P. D. (2014). An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research, 233*, 511–528.
Misir, M., Verbeeck, K., Causmaecker, P. D., & Berghe, G. V. (2011). An intelligent hyper-heuristic framework for CHeSC 2011. In *Learning and intelligent optimization – 6th international conference, LION 6, Paris, France, January 16–20, 2012, Revised Selected Papers*. In Y. Hamadi & M. Schoenauer (Eds.). *Lecture notes in computer science* (Vol. 7219, pp. 461–466). Springer.
Misir, M., Verbeeck, K., Causmaecker, P. D., & Berghe, G. V. (2013). An investigation on the generality level of selection hyper-heuristics under different empirical conditions. *Applied Soft Computing, 13*, 3335–3353.
Mumford, C. L. (2008). An order based memetic evolutionary algorithm for set partitioning problems. In J. Fulcher & L. C. Jain (Eds.), *Computational intelligence: a compendium*. *Studies in computational intelligence* (Vol. 115, pp. 881–925). Springer<http://dblp.uni-trier.de/db/series/sci/sci115.html>.
Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J. A., Walker, J., Gendreau, M., et al. (2012). Hyflex: a benchmark framework for cross-domain heuristic search. In *Proceedings of the 12th european conference on evolutionary computation in combinatorial optimization*. *EvoCOP'12* (pp. 136–147). Berlin, Heidelberg: Springer-Verlag. http://dx.doi.org/10.1007/978-3-642-29124-1_12.
Okano, H. (2002). A scanline-based algorithm for the 2D free-form bin packing problem. *Journal of the Operations Research Society of Japan, 45*, 145–161.
Pappa, G. L., Ochoa, G., Hyde, M., Freitas, A., Woodward, J., & Swan, J. (2013). Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 1–33.
Pillay, N. (2012). A study of evolutionary algorithm selection hyper-heuristics for the one-dimensional bin-packing problem. *South African Computer Journal*, 31–40.
Ponce-Pérez, A., Pérez-García, A., & Ayala-Ramírez, V. (2005). Bin-packing using genetic algorithms. In *Proceedings of the 15th international conference on electronics, communications and computers* (pp. 311–314). Washington, DC, USA: IEEE Computer Society. http://dx.doi.org/10.1109/CONIEL.2005.25.
Rodgers, J. L., & Nicewander, W. A. (1988). Thirteen ways to look at the correlation coefficient. *The American Statistician, 42*, 59–66.
Rohlfshagen, P., & Bullinaria, J. A. (2010). Nature inspired genetic algorithms for hard packing problems. *Annals of Operations Research, 179*, 393–419.
Ross, P. (2014). Hyper-heuristics. In E. K. Burke & G. Kendall (Eds.), *Search methodologies: introductory tutorials in optimization and decision support techniques* (2nd ed., pp. 611–638). New York: Springer.
Ross, P., Marín-Blázquez, J. G., Schulenburg, S., & Hart, E. (2003). Learning a procedure that can solve hard bin-packing problems: a new GA-based approach to hyper-heuristics. In *Conference on genetic and evolutionary computation. Lecture notes in computer science* (Vol. 2724, pp. 1295–1306). Springer-Verlag.
Scholl, A., Klein, R., & Jürgens, C. (1997). Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research, 24*, 627–645.
Sim, K., & Hart, E. (2013). Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In C. Blum & E. Alba (Eds.), *GECCO* (pp. 1549–1556). ACM.
Sim, K., Hart, E., & Paechter, B. (2012). A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, & M. Pavone (Eds.), *Parallel problem solving from nature - PPSN XII. Lecture notes in computer science* (Vol. 7492, pp. 348–357). Berlin Heidelberg: Springer.
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 *Conference on genetic and evolutionary computation* (pp. 591–598). New York, NY, USA: ACM Press.
Terashima-Marín, H., Ortiz-Bayliss, J. C., Ross, P., & Valenzuela-Rendón, M. (2008). Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. In *GECCO '08: proceedings of the 10th annual conference on genetic and evolutionary computation* (pp. 571–578). New York, NY, USA: ACM<http://doi.acm.org/10.1145/1389095.1389206>.
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*, 369–392.
Uday, A., Goodman, E. D., & Debnath, A. A. (2001). Nesting of irregular shapes using feature matching and parallel genetic algorithms. In Goodman, E. D. (Ed.), *Genetic and evolutionary computation conference. Late breaking papers* (pp. 429–434).
Vázquez-Rodríguez, J., Petrovic, S., & Salhi, A. (2007). An investigation of hyper-heuristic search spaces. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 3776–3783). doi: <http://dx.doi.org/10.1109/CEC.2007.4424962>.
Walker, J. D., Ochoa, G., Gendreau, M., & Burke, E. K. (2012). Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. In *Learning and intelligent optimization – 6th international conference, LION 6, Paris, France, January 16–20, 2012, Revised Selected Papers*. In Y. Hamadi & M. Schoenauer (Eds.). *Lecture notes in computer science* (Vol. 7219, pp. 461–466). Springer.
Wang, W. X. (1998). Binary image segmentation of aggregates based on polygonal approximation and classification of concavities. *Pattern Recognition, 31*, 1503–1524.
Wäscher, G., & Gau, T. (1996). Heuristics for the integer one-dimensional cutting stock problem: A computational study. *OR Spectrum, 18*, 131–144.
Wäscher, G., Hausner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research, 183*, 1109–1130 [Special issue on Cutting, Packing and Related Problems].