Soft Computing (2023) 27:12997–13010
https://doi.org/10.1007/s00500-022-07118-4

FOCUS

A hierarchical hyper-heuristic for the bin packing problem

Francesca Guerriero[1] · **Francesco Paolo Saccomanno**[1]

Accepted: 1 April 2022 / Published online: 07 May 2022

Abstract
This paper addresses the two-dimensional irregular bin packing problem, whose main aim is to allocate a given set of irregular pieces to larger rectangular containers (bins), while minimizing the number of bins required to contain all pieces. To solve the problem under study a dynamic hierarchical hyper-heuristic approach is proposed. The main idea of the hyper-heuristics is to search the space of low-level heuristics for solving computationally difficult problems. The proposed approach is "dynamic" since the low-level heuristic to be executed is chosen on the basis of the main characteristics of the instance to be solved. The term "hierarchical" is used to indicate the fact that the main hyper-heuristic can execute either simple heuristics or can run in a "recursive fashion" a hyper-heuristic. The developed solution strategy is evaluated empirically by performing extensive experiments on irregular packing benchmark instances. A comparison with the state-of-the-art approaches is also carried out. The computational results are very encouraging.

Keywords Packing · Irregular pieces · Hyperheuristic

1 Introduction

This paper addresses the two-dimensional irregular bin packing problem (2DIBPP), an NP-hard combinatorial optimization problem (Pandey 2021). The aim is to pack a set of irregular pieces, into homogeneous rectangular stock sheets (bins) of fixed dimensions, in such a way that the number of used bins is minimized, that is the space utilization is maximized.

On the basis of the topology proposed in Wäscher and Schumann (2007), the problem under study falls into the general class of the two-dimensional single bin-size bin packing problem. However, additional features are considered, that is all pieces have irregular convex shape, and they cannot overlap with each other and cannot be flipped.

The 2DIBPP arises in several real-life applications, where it is necessary to cut irregular shape pieces from multiple homogeneous rectangular sheets. It is commonly addressed in the ceramic tile sector, shipbuilding industry, clothing, footwear and furniture production, logistics as well as glass, sheet metal processing and automobile industries (Okano 2002; Martinez-Sykora et al. 2017; Hu et al. 2020; Bennell et al. 2018; Luo et al. 2022).

Despite its practical importance, given the need of dealing with complex geometry, the 2DIBPP has attracted little attention in the scientific literature compared to other bin packing problems, and the majority of the scientific contributions have been focused on heuristic approaches.

In what follows, a concise overview of the literature related to the 2DIBPP is provided. The attention is focused on the scientific contributions most relevant to our study.

State-of-the-art Hyper-heuristic methods, based on genetic algorithms, for solving two-dimensional regular (rectangular) and irregular (convex polygonal) have been presented in Terashima-Marín et al. (2010), Lopez-Camacho et al. (2013), López-Camacho et al. (2014).

In particular, in Terashima-Marín et al. (2010) the authors proposed several heuristics for selecting the pieces and the bins, and for placing the pieces into the bins. In addition, they built a benchmark set of 540 instances, used to evaluate the performance of the developed approaches. An extension of the Djang and Finch heuristics, originally designed for the one-dimensional bin packing problem, to the two-dimensional case has been proposed in Lopez-Camacho et al. (2013). Very good results have been achieved on convex poly-

Communicated by Dario Pacciarelli.

✉ Francesco Paolo Saccomanno
francescopaolo.saccomanno@unical.it

Francesca Guerriero
francesca.guerriero@unical.it

[1] DIMEG, University of Calabria, Rende, Italy

gon instances. An evolutionary selection and constructive hyper-heuristic approach that combines single heuristics has been proposed by the same authors in López-Camacho et al. (2014).

In Okano (2002), a scanline-based algorithm is proposed to approximate the irregular shape of the pieces and a First-Fit Decreasing (FFD) algorithm is used to address the placement problem.

All the aforementioned papers do not allow pieces rotation. This possibility is instead exploited in Martinez-Sykora et al. (2017), Abeysooriya et al. (2018), Liu et al. (2020) and Zhang et al. (2022). In particular, in Martinez-Sykora et al. (2017) the authors considered free rotations of pieces and presented several variants of a construction algorithm. In order to address both the pieces assignment and the pieces allocation problems, some integer programming models are formulated and solved. In Abeysooriya et al. (2018), it is assumed that both free rotations and limited rotations (i.e., 90°, 180°, 270°, 360°) are allowed and, unlike previous contributions, the allocation and placement problems are solved together, by means of a heuristic approach.

The variant of the 2DIBPP with limited rotations has been also studied in Liu et al. (2020). In particular, a constructive solution approach is proposed, in which the pieces are assigned to the bins by using the FFD strategy; the placement problem is solved by the bottom-left algorithm and the pieces exchange method. Furthermore, a greedy local search approach is executed to improve the solution quality.

An extension of this work is given in Zhang et al. (2022), where the authors introduce a waste least first decreasing strategy to assign the pieces to the bins, whereas an overlap minimization approach, based on the separation algorithm presented in Imamichi et al. (2009) and Leung Lin (2012), is used to address the placement problem. To improve the solution, a greedy local search approach that relies on pieces swapping between two bins is proposed.

Contribution and organization of the paper This work focuses on the development of a hyper-heuristic approach to solve the 2DIBPP. The hyper-heuristic is used to define a high-level heuristic, which controls low-level heuristics. In particular, it should decide when and where to apply each single low-level heuristic, depending on the characteristic of the given problem state.

There are two main types of hyper-heuristics (Drake et al. 2020), categorized by whether they are used for selecting existing heuristic (base low-level heuristic) or generating new heuristics from components of above heuristic. The approach presented in this paper is of the first type. The aim is to choose base low-level heuristics, which allow to select and place the pieces into the bins in an efficient way.

The main contributions of this paper are the following:

- development of a novel hyper-heuristic, able to provide better results than those of the state-of-the-art;
- implementation of a benchmark system that exploits information related to the behavior of each low-level heuristic in terms of solution quality, to improve the performance of the hyper-heuristic in a dynamic fashion;
- design and implementation of innovative and efficient feature-based selection and placement heuristics;
- development and testing of a parallel version of the proposed hyper-heuristic.

The rest of the paper is organized as follows. Section 2 describes the main characteristics of the 2DIBPP addressed in this work. The proposed low-level heuristics are presented in Sect. 3, whereas the developed solution approach is detailed in Sect. 4. Section 5 is devoted to the discussion on the computational results. The paper ends with some concluding remarks given in Sect. 6.

2 Problem description

In this section, a formal description of the specific variant of the 2DIBPP addressed in this paper is provided.

Let $P = \{p_1, p_2, \ldots, p_n\}$ denote the set of n irregular pieces. Each piece $p_k, k = 1, \ldots, n$ is represented as a polygon, whose area is denoted as a_k. It is assumed that pieces rotations and flipping are not allowed.

Let $B = \{b_1, b_2, \ldots, b_N\}$ denote the set of N identical rectangular bins, which can be used to pack the n pieces. Each bin b_i, $i = 1, \ldots, N$ is a rectangular object, characterized by a fixed width W and fixed length L. It is assumed that the number of available bins N is large enough to contain all n pieces.

The objective of the 2DIBPP is to minimize the number of bins used to pack all the n pieces, by ensuring that:

- the pieces are placed entirely within the bin;
- the pieces do not overlap with each other.

It is worth observing that the number of bins used to pack the n items does not allow to discriminate among solutions characterized by the same number of bins. For this reason, the following two functions are considered to evaluate a solution.

The first one is the fitness function F, introduced in Lopez-Camacho et al. (2013), defined as follows:

$$F = \frac{\sum_{i=1}^{N} U_i^2}{N};$$

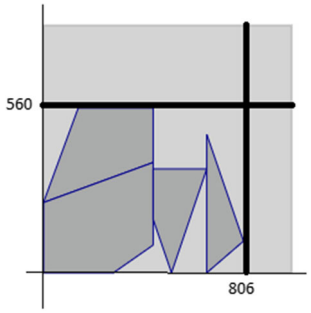

Fig. 1 The value of K corresponding to a solution for the instance TA009, for which $N = 4$; $K = 4 - 1 + 0.56 = 3.56$

where U_i represents the ratio utilization of each bin b_i, $i = 1, \ldots, N$, that is:

$$U_i = \frac{\sum_{k=1}^{n_i} a_k}{L \times W};$$

where n_i represents the number of pieces placed in the bin b_i, $i = 1, \ldots, N$.

The other evaluation measure is the fractional number of bins K, introduced in Han et al. (2013) and Martinez-Sykora et al. (2017) to take into account the reusable part of a bin and relies on the use of a cut, either vertical or horizontal, which recovers the final non-utilized part of a bin (see Fig. 1). The parameter K is defined as follows:

$$K = N - 1 + R^*;$$

where R^* represents the portion of the bin used once the reusable residual area has been removed by applying the vertical (horizontal) cut.

In this paper, the fitness function F is used to guide the proposed hyper-heuristic to explore the solution space, whereas the K is used to evaluate the performance of the proposed solution strategy in comparison with the state-of-the-art approaches.

3 Low-level heuristics

The proposed low-level heuristics rely on the combination of "heuristic operators," which are essentially of the following three types: selection, placement and filling operators.

They offer basic functionalities for the exploration of the solution space. Thus, the developed hyper-heuristic is used to find an appropriate combination of these operators, with the aim of building good-quality solutions. The selection operators are used to extract the pieces to be allocated into the containers, by taking into account some specific features associated with the pieces (see Sect. 3.1). More specifically, the next piece to be handled is chosen by considering the presence or absence of these features, according to a certain cutoff value, experimentally determined.

The extracted piece is then placed in the best position by applying a placement operator. Four different operators have been implemented: Bottom Up (PBU), Top Down (PTD), Left Right (PLR) and Right Left (PRL).

In particular, in PBU the pieces are positioned from bottom to top in columns (Bottom Up), in PTD from top to bottom always in columns (Top Down), in PLR from left to right proceeding by rows (Left Right), and finally, in PRL from right to left (Right Left) always in a row. Recursive combinations of these four operators were also considered. Finally, the filling operators are used to fill the empty spaces according to a greedy strategy.

In what follows, the defined heuristic operators are described in details.

3.1 Selection operators and features

This section presents the main selection operators that are used to extract the pieces to be placed, by both the placement and the filling operators. It is worth observing that other selection strategies can be derived, by combining the main operators in an appropriate way.

For each piece, the following features have been defined (see Fig. 2):

HorizontalityDown: it represents the ratio between the width of the base (the lowest horizontal segment of the piece with slope = 0) and the total width of the piece; in the example of Fig. 2, it is equal to 66%.
HorizontalityTop: as above, but it takes into consideration the upper segment;
VerticalityRight: it indicates the ratio between the vertical right side of the piece (the one with slope = infinite) and the overall height of the piece; in the example of Fig. 2, it is equal to 0%,
VerticalityLeft: as above, but for the left side;
HorizontalityDelta (VerticalityDelta): it represents the delta between the two extreme points of the figure, and this parameter is very useful for choosing pieces to be put together;
Area: Area of the piece;
Barycenter: The barycenter of the piece.

The main aim of the proposed operators is to select in a "smart way" the pieces to be placed, by using as criteria the features introduced above.

Since the outcome depends on the placement strategy used, the pieces already inserted in the bin and the remaining area, each selection operator is defined in such a way to work in symbiosis with the applied placement operator. The selection operators behave differently depending on whether

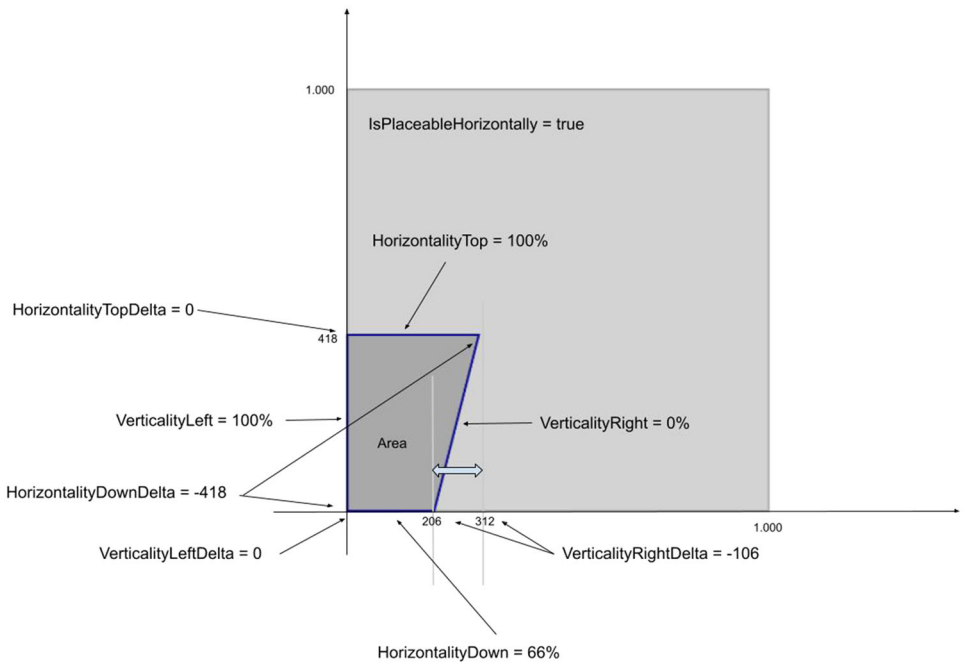

Fig. 2 Graphical representation of the features

they are used to individuate the first piece or the following ones.

If the solution is built by using the PBU placement strategy, the selection operator (referred to as *bottom-up selection operator*) handles the first pieces to be selected, in descending order of the HorizontalityDown, VerticalityLeft and Area features, while for the subsequent ones, the operator gives priority to the pieces that can be placed side by side, i.e., the pieces for which the difference between the HorizontalityDelta feature is less than a given threshold parameter.

In the case in which the PLR placement algorithm is used, the corresponding selection operator (i.e., *left-right selection operator*) works in similar way: in the case of the first piece, it gives priority to the same features HorizontalityDown, VerticalityLeft, Area as the bottom-up version. In the case of subsequent pieces, it selects the pieces in descending order of the VerticabilityLeft feature, since the aim is to select pieces that can be placed side by side on the current row.

3.2 Filling operators

This section describes the proposed filling operators, whose main aim is to place the pieces within the available space, using several heuristics, which are guided by local fitness functions.

The proposed operators can be viewed as grouped into three main classes (referred to as FA, FB and FC), which differ from each other for the specific fitness function used. Other operators can be derived, by modifying, in an appropriate way, the relevant parameters.

The FA operator inserts pieces into the bin, trying to optimize the position of their barycenter.

In particular, a fitness function is used to determine the coordinate (X, Y) of the barycenter of each piece. The highest priority is assigned to the pieces with the minimum value of the X coordinate, which are then selected first and, in the case of a tie, the pieces with the minimum value of the Y coordinate are processed.

More specifically, the first piece is placed at the bottom left. For the subsequent pieces, the positions tested are all and only those corresponding to the vertices of the area formed by the pieces already placed in the bin. If the vertices are too close to each other (for example when the polygon represents the approximation of a circle), these are discarded, and if they are too sparse, a virtual one is added. It is worth observing that considering only the vertices allows to improve the efficiency of the algorithm, but a deterioration in the solution quality could be observed. The proposed strategy is referred to as virtual vertex search (VVS), since the main idea is to either reduce or virtually increase the vertices to be considered, depending on whether they are too dense or too sparse.

The VVS differs from the no-fit polygon approach, generally used in the scientific literature (Dean et al. 2006; Domovic et al. 2014) since in the latter all possible placements of a polygon are evaluated.

Figure 3 reports the configuration obtained by applying the FA operator. More specifically, for each vertex of the area made up of the pieces already placed, the operator tries to place the new one and then check if some constraints (i.e., the piece must be inside the bin and must not overlap with the already inserted pieces) are satisfied; in the affirmative

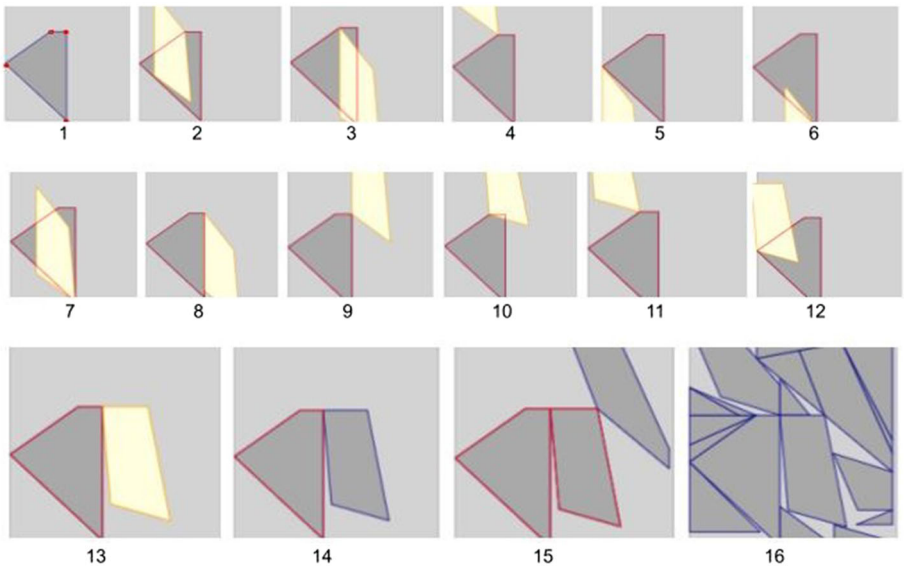

Fig. 3 Graphical representation of the configurations obtained by applying the filling operator FA. Configurations 1, 13, 14, 15, 16 are feasible

case, it verifies whether the position is improved by using the fitness function described above. With reference to the example given in Fig. 3, the pieces are outside in 2, 3, 4, 5, 6, 8, 9, 10, 11, 12; the pieces overlap in configuration 7.

While the FA heuristics processes the pieces sorted in decreasing order of the area and checks all the vertices where it is possible to place the pieces, the FB operator instead tries to handle the pieces using a different fitness function. More specifically, it calculates the distance between the vertices of the piece to be placed and those of the polygon corresponding to the pieces already placed (i.e., the occupied area). If the determined distance is less than a predefined cutoff value (i.e., ϵ), the fitness function is increased by $\frac{1}{distance}$ (i.e., $Fitness := Fitness + \frac{1}{distance}$). The piece for which the fitness function assumes the minimum value is selected.

The last filling operator (i.e., FC) is based on the same fitness function used in FB. Its purpose is not to find a solution for the current bin, but to create clusters of pieces as compact as possible and with a fairly rectangular shape, which will be handled as single larger pieces.

3.3 Placement operators

This section gives a detailed description of the considered placement operators (that is, PBU, PTD, PLR, PRL). As mentioned above, the PBU (PTD) places the extracted piece from bottom to up (from top to down) in columns, whereas in PLR (PRL) the piece is placed from left to the right (from right to left) proceeding by rows.

In what follows, the attention will be focused on the PBU and the PLR operators.

For the sake of comprehension, it is useful to consider the example reported in the upper box of Fig. 4, which gives the solution obtained by applying the PBU placement operator

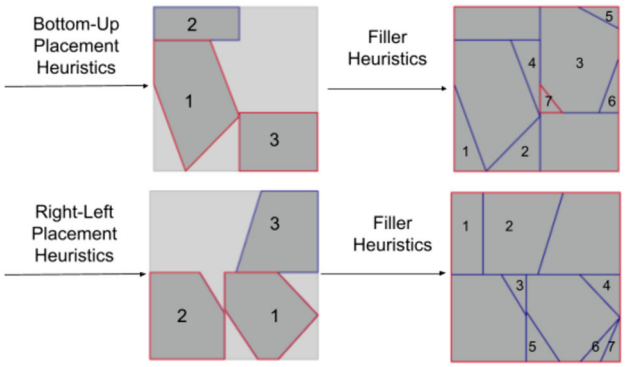

Fig. 4 The Placement Heuristics inserts the first pieces according to the bottom/up strategy (in the upper box) or according to the right/left strategy (in the lower box). Then, the Filler Heuristics inserts the pieces, sorted by size, in the available spaces as long as possible

with the corresponding selection operator and the FA filler operator.

The PBU operator works by columns from bottom to up: in the first step it receives, from the selection operator, a piece that has the maximum height among all the pieces that have the greatest value of the HorizontabilityDown and VerticabilityLeft features.

This allows to extract the best piece to be placed in the lower left corner (indicated in Fig. 4 with the number 1). Successively, the selection operator is used to individuate a piece with the width as much as possible equal to the one just inserted and with HorizontabilityDown > 0, in order to be sure that the two pieces can be placed side by side (the piece numbered with 2 in the upper box of Fig. 4). The subsequent column is then considered and the approach tries to find the piece (numbered with 3 in Fig. 4), which occupies as much as possible the residual width. Since, after this case,

it is not possible to find other pieces to be inserted above the one selected, the placement heuristic stops and as the columns are finished, the hyper-heuristic executes the filling operators, which fill the empty spaces, by selecting the pieces in decreasing order of the area.

The right-left placement operator works in a similar manner, but instead of operating by columns and from bottom to up, it operates by rows from right to left (see the lower box of Fig. 4).

Even though the placement operators are very simple, their combination allows to find different effective solutions and the obtained results are comparable with those of the state-of-the-art. This occurs especially when the area occupied by the pieces is commensurated with the area of the bin (as explained in Sect. 5).

4 The hyper-heuristic approach

In this section, the hyper-heuristic developed to address the 2DIBPP is described in details.

The proposed solution strategy relies on the execution of the low-level heuristics presented in the previous section. These operators have been defined by considering some basic elements derived from the solution strategies proposed in the scientific literature and are executed and combined in different ways, by the main hyper-heuristic approach. Thus, the main algorithm is not used to build a solution, but instead the aim is to select, in a hyper-heuristic way, these heuristic operators in order to obtain a good-quality solution.

It is worth observing that the basic operators behave differently, depending on the characteristics of the instance to be solved. Thus, in order to fully exploit the capability of the developed operators, since for some instances the mere combinations could not provide good-quality solutions, a *hierarchical* hyper-heuristic approach has been defined (detailed in Sect. 4.1).

The main idea is to divide the instance into blocks, to which the heuristic operators are applied recursively. Thus a hierarchical approach is obtained: the main hyper-heuristic relies on the execution of lower-level hyper-heuristics, applied on a single block.

In the proposed approach, the heuristic operators always work on only one bin at a time; indeed, it is not allowed the possibility of exchanging pieces between bins.

This choice was made to allow the algorithm to work even in parallel, i.e., the hyper-heuristic can explore in parallel the search space of the basic heuristics, in order to speed up the overall algorithm. This possibility is not considered in the algorithms proposed in the scientific literature, since generally the solution strategy either takes into consideration all the bins or, as a last step, performs a local search, by exchanging pieces between different bins. The flow diagram of the developed approach is depicted in Fig.5, from which it is evident that, until there are pieces to be processed, the following operations are executed:

- A new bin is initialized and the low-level heuristics are loaded to be executed;
- Several combinations of different low-level heuristics in the current bin (sequentially or in parallel) are evaluated;
- The solution, which provides the best value, is selected and the pieces are placed into the current bin, according to the selected configuration.

4.1 The hierarchical heuristic scheme

In order to improve the performance of the placement operators, when small pieces need to be handled, a "hierarchical heuristic" has been devised.

The main idea is to divide the bin area into blocks and to apply to each of them the heuristic operators, in a similar way to the main hyper-heuristic (Lopez-Camacho et al. 2013). Two different versions of the hierarchical heuristic have been defined, referred to as HHA and HHB.

HHA builds a solution by recursively applying placement operators. Figure 6 provides a graphical representation of the main operations executed by HHA.

The approach starts by considering an empty bin. In the first step (phase 1), the hyper-heuristic decides whether and which low-level heuristic operators to apply, by considering also the dimensions and the residual space of the bin (initially empty).

In the second step (phase 2), the previously selected low-level heuristics are executed and, thus, some initial solutions are determined. For each of them, the residual bound box is calculated and, therefore, if in the previous phase at least one piece has been inserted, the placement operators are applied recursively again (return to phase 1).

Finally, when the placement operators are not able to insert other pieces, due to the limited available free space, the filling operators are executed on the current solutions (phase 4) and the best one is selected.

The second variant of the proposed hierarchical heuristic (referred to as HHB) can be viewed as an enhanced version of HHA.

More specifically HHB, similarly to HHA, divides the space into blocks, but in addition, it executes a pre-composition procedure, whose main aim is to combine the pieces in a block of fairly rectangular shape. These sub-optimal solutions are determined by using the FC operator and considered in the hyper-heuristic as single pieces to be inserted in the bins.

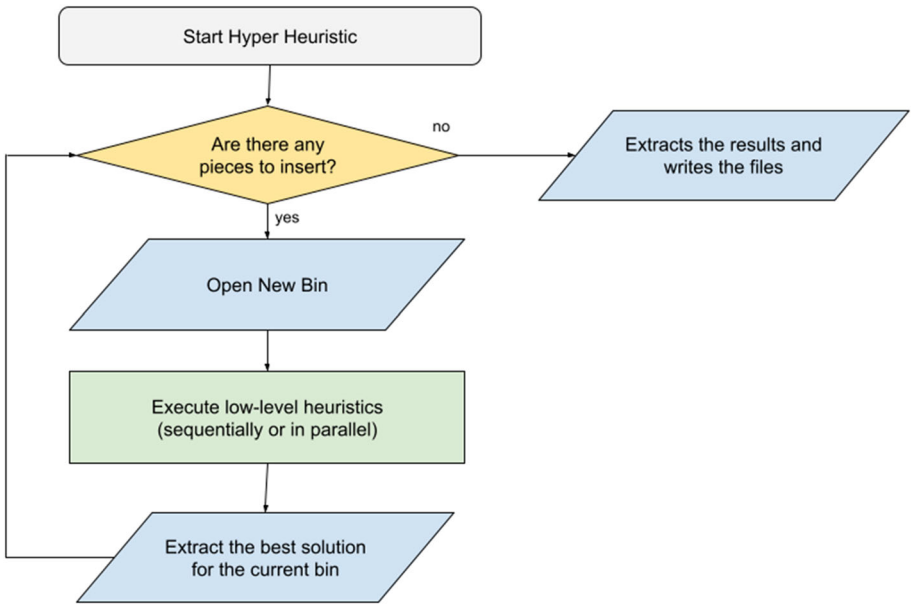

Fig. 5 Flowchart of the main algorithm

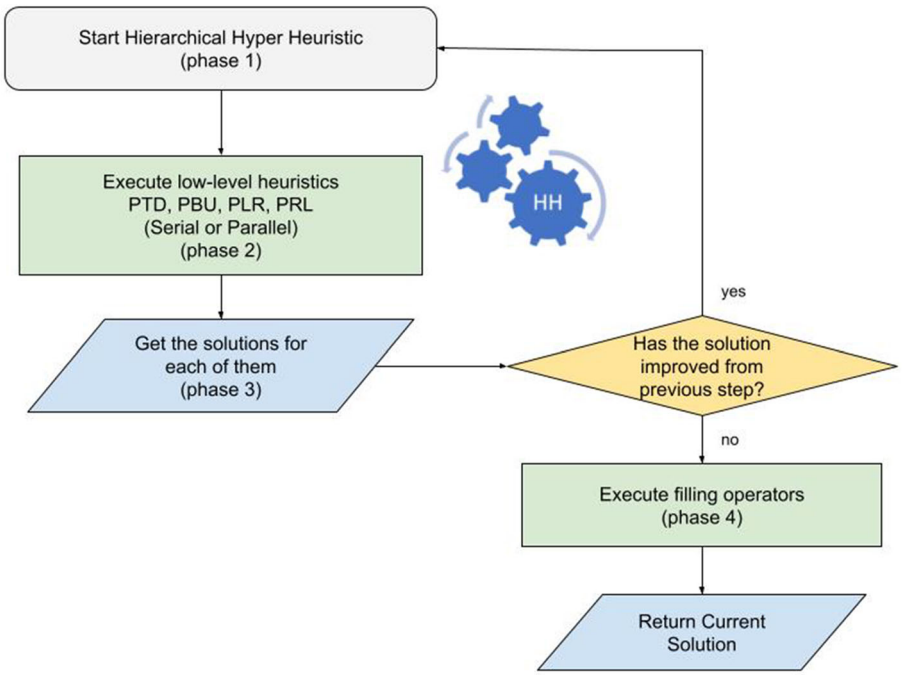

Fig. 6 Graphical representation of the hierarchical hyper-heuristic

4.2 Sequential and parallel versions

In the sequential version of the approach (Fig. 7), the order in which the low-level heuristics are executed plays an important role, since if an optimal solution is obtained for the current bin (i.e., the bin is fully filled), the other operators are not executed, with a consequent reduction in the total execution time.

Several approaches can be used to decide which low-level heuristic should be applied at each iteration. In the present work, the order has been determined experimentally. In particular, to empirically determine the order in which these low-level heuristics should be executed, a benchmark system has been developed that reports, for each bin, the combination of the basic operators that provided the best results. Then, the best performing heuristics are selected to be executed first.

More specifically, a priority queue, initialized on the basis of the specific features (Lopez-Camacho et al. 2013) possessed by the pieces to be placed, was created. The priority value assigned to each low heuristic gives a measure of the quality of the solutions determined so far. This value is

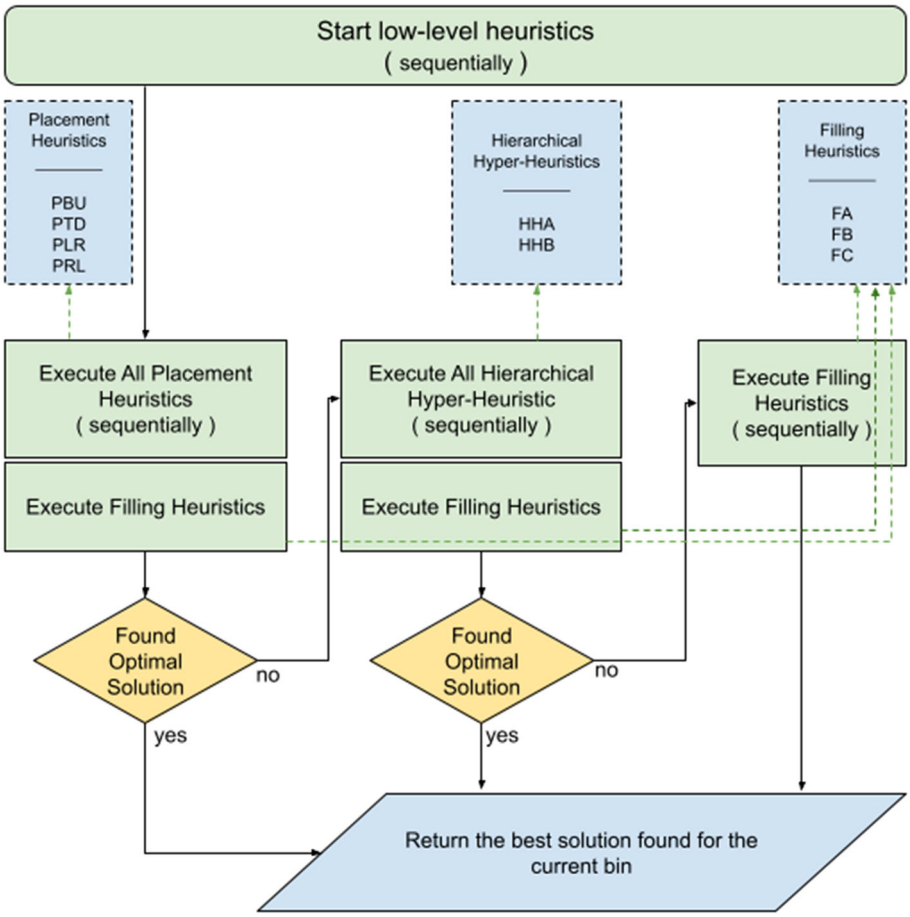

Fig. 7 Flowchart of the sequential version of the algorithm

updated dynamically on the basis of a reinforcement learning mechanism (Dargan et al. 2020; Dong et al. 2021).

In particular, each time a heuristic is executed and a solution is determined, it receives a reward proportional to the fitness value obtained and the corresponding priority value is updated. This priority scheme is embedded into the hyper-heuristic, and it is used to select the low-level heuristic to be executed at each iteration.

In the parallel version (Fig. 8), all basic heuristics are performed in parallel. In particular, all types of placement heuristics, all hierarchical hyper-heuristic and all filling operators. Each of these heuristics runs in parallel on a separate thread. If an optimal solution is found (e.g., all bins are filled up to 100%), this solution is returned immediately and the other threads are stopped. In the general case, however, all the low-level heuristics are performed and the most effective solution is selected upon completion.

5 Computational experiments

This section presents the computational results, collected by testing the proposed solution strategies. The developed algorithms have been implemented in the c# language, by using visual studio 2019 as development environment. The experiments have been carried out on a PC with a core i7 processor and 16 GB memory.

The computational experiments have been designed with the aim of assessing the performance of the single operators and their combination in the hyper-heuristic framework. A comparison with the state-of-the-art (i.e., results reported in Lopez-Camacho et al. (2013), Martinez-Sykora et al. (2017), Liu et al. (2020)) is also carried out.

The testings have been conducted on the benchmarks proposed in Terashima-Marín et al. (2010), which are grouped into 18 different sets ($A \ldots R$), each of them containing 30 instances randomly generated. The characteristics of these 540 instances are given in Table 1, where, for each type, the object size, the number of pieces and the optimal solution are reported. It is worth observing that an optimal solution is determined only when all bins are filled up to 100%. From Table 1, it is evident that an optimal solution is available for all the instances, with the only exception of those of type G.

The pieces to be placed are made up of irregular convex polygons, with a number of sides between 3 and 8. The bins are square-shaped. These pieces have been studied in Lopez-

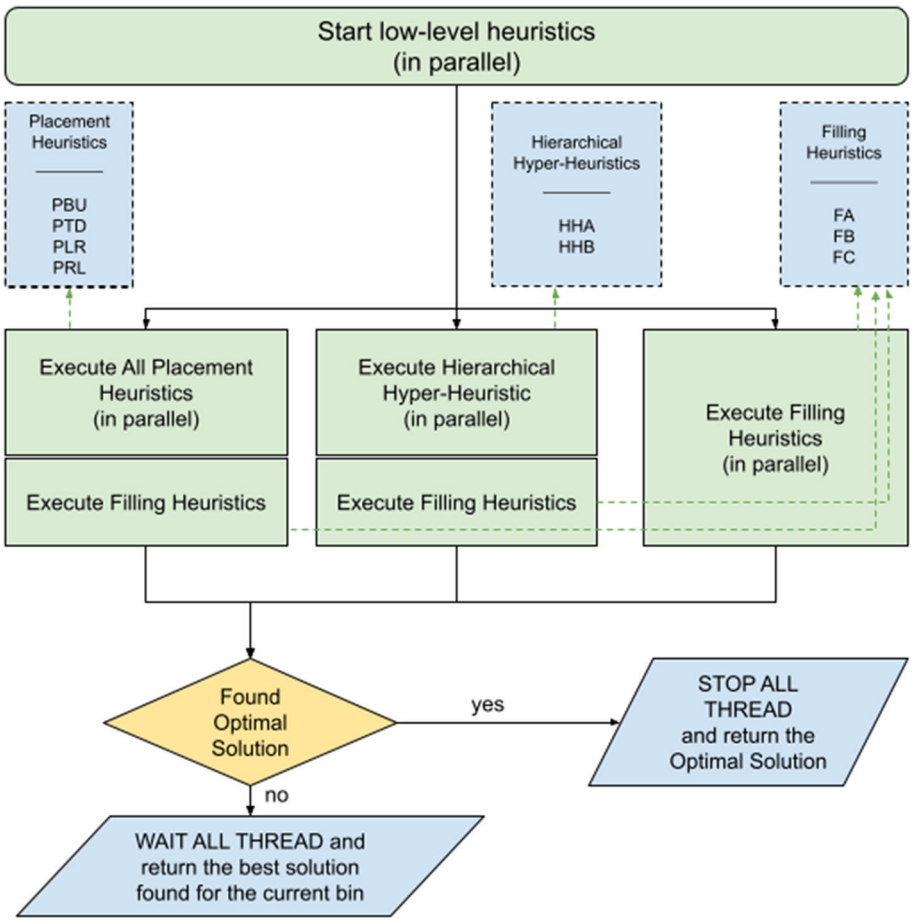

Fig. 8 Flowchart of the parallel version of the algorithm

Table 1 Test problem characteristics

Type	Objects Side	Pieces	Optimal (number of bins)
A	1000	30	3
B	1000	30	10
C	1000	36	6
D	1000	60	3
E	1000	60	3
F	1000	30	2
G	1000	36	≤ 15
H	1000	36	12
I	1000	60	3
J	1000	60	4
K	1000	54	6
L	1000	30	3
M	1000	40	5
N	1000	60	2
O	1000	28	7
P	1000	56	8
Q	1000	60	15
R	1000	54	9

Camacho et al. (2013), and the values of some features (i.e., size, rectangularity, etc.), reported in Table 2, have been calculated .

The performance of the proposed approaches has been evaluated by considering the following criteria:

- the average number of used bins N;
- the percentage usage of each bin F (see Sect. 2);
- the average fractional number of bins K (see Sect. 2);
- the execution time T in seconds.

In order to evaluate the gain obtained by combining the developed operators in the proposed hyper-heuristic framework, in a first phase of the computational experiments, the heuristic procedures were tested separately. The related results are given in Tables 3–5. In particular, Table 3 contains the results obtained using the placement strategies (i.e., PBU, PTD, PLR, PRL) combined with the corresponding selection and filling operators; the results provided by the hierarchical heuristics (HHA and HHB) are reported in Table 4, whereas Table 5 gives the computational results obtained by the filling operators (FA, FB and FC).

Table 2 Characteristics of test problems reported in Lopez-Camacho et al. (2013)

	Average Piece Area	Piece area Standard Deviation	Average Rectangularity	Percentage of right Angles	Percentage of vertical/horizontal Sides
Minimum	0.033	0.01	0.35	11	34
Total average	0.154	0.1	0.68	42	65
Maximum	0.35	0.28	1	100	100
Average of instance per type					
A	0.1	0.069	0.7	42	68
B	0.333	0.162	0.87	67	84
C	0.167	0.124	0.68	36	63
D	0.05	0.036	0.57	23	51
E	0.05	0.035	0.41	12	38
F	0.067	0.05	0.59	29	57
G	0.332	0.156	0.87	67	83
H	0.333	0.158	0.86	67	83
I	0.053	0.017	1	100	100
J	0.067	0.034	0.83	68	83
K	0.154	0.15	0.63	34	60
L	0.1	0.075	0.51	23	50
M	0.125	0.102	0.55	28	55
N	0.033	0.024	0.62	32	60
O	0.25	0.223	0.57	27	58
P	0.143	0.173	0.49	18	43
Q	0.25	0.053	0.89	51	76
R	0.167	0.153	0.63	39	62

Table 3 Computational results obtained with the placement strategies PBU, PTD, PLR and PRL

Instance type	N	F	K	T
A	4.000	0.620	3.133	5.353
B	10.067	0.992	10.033	0.350
C	7.033	0.795	6.267	2.905
D	4.800	0.461	3.800	34.799
E	5.367	0.366	4.400	17.098
F	3.100	0.456	2.333	5.540
G	14.400	0.679	13.400	2.748
H	12.000	1.000	12.000	0.380
I	3.933	0.649	2.900	21.521
J	5.067	0.657	4.100	26.793
K	7.467	0.716	6.500	3.609
L	4.700	0.492	3.767	2.678
M	6.767	0.610	5.833	3.436
N	3.067	0.475	2.100	56.081
O	7.167	0.969	6.933	0.415
P	10.467	0.652	9.533	5.761
Q	15.233	0.978	15.067	3.658
R	10.567	0.779	9.667	4.292
Average	7.511	0.686	6.765	10.968

Table 4 Computational results obtained with the hierarchical approaches HHA and HHB

Instance type	N	F	K	T
A	3.700	0.744	2.867	3.044
B	10.633	0.916	9.900	1.047
C	6.400	0.919	6.033	1.601
D	4.500	0.536	3.533	16.772
E	5.367	0.419	4.367	4.730
F	2.800	0.620	1.967	2.614
G	15.133	0.615	14.133	1.562
H	12.633	0.929	12.033	1.451
I	3.900	0.578	2.900	39.560
J	5.200	0.618	4.200	16.358
K	6.933	0.813	6.133	1.730
L	4.200	0.651	3.467	1.118
M	6.633	0.680	5.833	1.672
N	2.967	0.468	1.967	50.506
O	7.267	0.956	6.633	0.606
P	10.467	0.674	9.467	3.060
Q	15.133	0.988	15.000	5.083
R	10.167	0.838	9.467	3.423
Average	7.446	0.720	6.661	8.663

Table 5 Computational results obtained with the filling operators FA, FB and FC

Instance type	N	F	K	T
A	4.500	0.500	3.533	0.960
B	12.000	0.758	11.167	0.418
C	8.600	0.559	7.633	0.842
D	5.033	0.397	4.033	9.625
E	5.933	0.333	4.933	5.322
F	3.567	0.376	2.567	1.366
G	15.467	0.598	14.467	0.320
H	14.500	0.738	13.533	0.239
I	4.033	0.588	3.033	4.792
J	5.867	0.525	4.867	5.774
K	8.133	0.626	7.133	0.774
L	5.300	0.417	4.333	0.496
M	7.733	0.492	6.733	0.879
N	3.133	0.442	2.133	18.401
O	7.567	0.909	7.167	0.179
P	11.067	0.632	10.133	1.218
Q	18.233	0.700	17.367	1.065
R	12.067	0.619	11.067	1.598
Average	8.485	0.567	7.546	3.015

The numerical results given in Table 3 clearly underline that the placement strategies allow to obtain the optimal solutions only for the instances of type H. In addition, they showed a good behavior when solving instances of types B, O and Q, for which the value of the criterion F is greater than 0.96. Very poor performances ($F \leq 0.5$) are observed on 5 out of 18 types of instances (i.e., D, E, F, L and N). Thus, an average fitness value of 0.686 is obtained. The placement strategies are very efficient, requiring an average execution time of 11 seconds.

From Table 4, it is evident that the hierarchical heuristics outperform the placement strategies in terms of both solution quality and efficiency. Indeed, the average F increases to 0.720 and the average execution time reduces to about 9 seconds.

The results reported in Table 5, related to the filling operators, highlight that these operators are very efficient, but they are not able to bring satisfactory results in terms of solution quality. Indeed, an average value of F equal to 0.567 is obtained, whereas the average execution time is equal to 3.015 seconds. Thus, these operators allow to obtain reasonable results only if they are used in combination with the other strategies.

The computational results obtained by executing a combination of the operators in the order defined by the proposed hyper-heuristic are reported in Table 6.

Table 6 Computational results obtained with the hyper-heuristic

Instance type	N	F	K	T
A	3.367	0.869	3.000	6.447
B	10.000	1.000	10.000	0.356
C	6.133	0.970	5.933	3.758
D	4.100	0.602	3.200	62.658
E	4.467	0.538	3.500	24.904
F	2.733	0.648	2.067	8.680
G	14.133	0.711	13.167	3.770
H	12.000	1.000	12.000	0.386
I	4.000	0.680	3.000	82.287
J	5.000	0.690	4.100	47.361
K	6.400	0.923	5.967	4.028
L	3.533	0.806	3.000	3.308
M	5.533	0.873	5.033	3.525
N	3.000	0.507	2.033	117.646
O	7.033	0.994	6.967	0.447
P	9.000	0.847	8.233	6.922
Q	15.000	1.000	15.000	3.681
R	9.400	0.946	9.033	5.410
Average	6.935	0.811	6.402	21.421

Table 7 Average number of the times in which each operator has obtained the best result

PTD	PBU	PLR	PRL	HH
209	1976	199	389	529

The results reported in Table 6 underline that the proposed hyper-heuristic outperforms the single operators in terms of solution quality; on the other hand, the combined approach is more time-consuming. Indeed, for instances of type B, H and Q, it is able to find the optimal solutions (100% filling of the bins); in addition, for other four types of instances the value of F is greater than 0.9 (i.e., C, K, O and R). The computational overhead increases to 21.421 seconds on average.

In order to assess the behavior of the implemented operators in terms of solution quality, Table 7 gives the number of times in which each operator has determined the best solution, on the considered instances. From Table 7, it is evident that the best performance is obtained by PBU, which has returned the best solution 1976 times, followed by HH, PRL, PTD and PLR.

In Table 8 the performance of the proposed hyper-heuristic is compared with the state-of-the-art approaches presented in Lopez-Camacho et al. (2013), Martinez-Sykora et al. (2017), Liu et al. (2020)). It is important to point out that the results of Table 8 are obtained with the sequential version of the proposed approach. In addition, in order to ensure a fair comparison in terms of computational overhead, the experiments

Table 8 Comparison with the state-of-the-art approaches

Instance type	Lopez-Camacho et al. (2013)	Martinez-Sykora et al. (2017)					Liu et al. (2020)	Our Approach			
	F	N	F	K	T	F with LS2	F	N	F	K	T
A	0.605	4	0.614	3.49	51.7	0.614	0.632	3.367	0.869	3.000	27.128
B	0.929	11.33	0.826	11.1	27.83	0.878	0.880	10.000	1.000	10.000	0.955
C	0.763	7.233	0.731	6.838	18.2	0.736	0.797	6.133	0.970	5.933	7.593
D	0.579	4	0.591	3.638	149.5	0.590	0.610	4.100	0.602	3.200	100.323
E	0.412	4	0.523	4.098	100.5	0.529	0.540	4.467	0.538	3.500	44.069
F	0.496	3	0.516	2.412	39.733	0.564	0.548	2.733	0.648	2.067	16.269
G	0.814	14.467	0.724	14.05	3.3	0.809	0.819	14.133	0.711	13.167	12.689
H	0.928	14	0.786	13.693	66.267	0.810	0.883	12.000	1.000	12.000	1.103
I	0.627	4	0.629	3.383	87.067	0.652	0.697	4.000	0.680	3.000	130.966
J	0.665	5	0.672	4.57	90.533	0.701	0.698	5.000	0.690	4.100	80.774
K	0.718	7.033	0.761	6.721	51.533	0.763	0.768	6.400	0.923	5.967	8.675
L	0.512	4.067	0.589	3.751	28.467	0.619	0.630	3.533	0.806	3.000	7.441
M	0.589	6.4	0.658	6.102	24.2	0.655	0.723	5.533	0.873	5.033	8.911
N	0.503	3	0.518	2.388	199.7	0.668	0.549	3.000	0.507	2.033	191.743
O	0.823	7.3667	0.929	7.2584	10.567	0.848	0.937	7.033	0.994	6.967	1.247
P	0.678	9.8	0.72	9.383	53.967	0.852	0.717	9.000	0.847	8.233	14.094
Q	1.000	15.5	0.956	15.263	7.8	0.965	0.956	15.000	1.000	15.000	7.826
R	0.771	10.567	0.767	10.165	39.967	0.767	0.786	9.400	0.946	9.033	11.064
Average	0.690	7.487	0.695	7.128	58.380	0.723	0.732	6.935	0.811	6.402	37.382

have been carried out on a PC with an Intel i5-8250 processor and 8 GB memory.

Table 8 highlights that the average fitness obtained in Lopez-Camacho et al. (2013) is equal to 0.69. Successively in 2017, Martines et al. (Martinez-Sykora et al. 2017) obtained better results. Indeed, an average fitness of 0.695 has been reached. This value has been improved to 0.723 in the same work by applying a local search algorithm, at the expense of the average execution time which reached 161 seconds. Better results have been obtained in Liu et al. (2020), where an average fitness of 0.732 has been determined, with an average execution time of 75 seconds.

On average, the proposed hyper-heuristic outperforms the state-of-the-art approaches both in terms of solution quality and efficiency. Indeed, an average fitness equal to 0.811 has been obtained, while keeping the average run time equal to about 38 seconds.

In order to investigate how the performance of the proposed approach is affected by the instance characteristics, it is useful to consider the features of the pieces, described in Lopez-Camacho et al. (2013) and reported in Table 2.

From the collected results, it is evident the implemented solution strategy behaves the best over than 60% of the types of instances.

In the remaining instances, comparable results have been obtained, with a slight deterioration of the performance for the instances for which the average area of the pieces is less than 0.05 and, therefore, the number of pieces is very high (i.e., 60). Table 9 reports the computational times of the sequential version and the parallel counterpart, using 21 threads.

The collected results underline that on average the parallel version is more efficient than the sequential counterpart. In particular, an average reduction of the 50% in the execution time has been observed. However, the advantage of using the parallel algorithm is more evident for those instances, which require relevant computational time. Indeed, in the easiest instances, the overhead due to the parallel algorithm setup hinders the advantages of the parallel approach.

Table 9 Average execution time of the parallel and the sequential versions of the proposed approach

Instance Type	T (Sequential)	T (Parallel)
A	6.447	2.341
B	0.356	0.827
C	3.758	2.145
D	62.658	22.634
E	24.904	10.022
F	8.68	3.457
G	3.77	1.521
H	0.386	1.340
I	82.287	56.522
J	47.361	17.809
K	4.028	2.625
L	3.308	1.438
M	3.525	2.229
N	117.646	48.533
O	0.447	0.787
P	6.922	3.754
Q	3.681	4.922
R	5.41	4.151
Average	21.421	10.392

6 Conclusions

In this paper, the two-dimensional irregular bin packing problem has been addressed. In particular, a hyper-heuristic approach has been developed and tested on benchmark instances. The proposed approach relies on the use of very simple low-level heuristics and exploits a hierarchical strategy, where the hyper-heuristic is executed recursively. The behavior of the proposed approach has been empirically tested, on benchmark instances. The obtained results are very encouraging: on average, the proposed solution strategy outperforms the state-of-the-art approaches in terms of both solution quality and efficiency. The present work has shown that simple low-level heuristics, which are not able to find good-quality solutions, if combined in a hyper-heuristic allow to find solution of good quality, in a limited amount of time.

The proposed approach allows to manage one bin at a time. This choice has reduced the possibility of implementing local search moves that consider the possibility of exchanging pieces between bins and to obtain better solutions. However, on the other hand, it was possible to define a parallel version of the approach, with a consequent reduction in the computational time.

The solution approach proposed in this work allows to address the bin packing problem when convex pieces are considered and the pieces cannot be rotated and flipped. The development of enhanced versions of the proposed hyper-heuristic defined in such a way to handle the variants of the problem not considered in this work (i.e., non-convex, rotated and flipped pieces) will be the subject of future research.

Acknowledgements The authors would like to thank the anonymous referees and the editor for their constructive comments, which have significantly improved the presentation of this paper. Furthermore, they would like to acknowledge the contribution of the professionals at EuroPan S.r.l.

Author Contributions Both the authors contributed to the design and implementation of the research, to the analysis of the results and to the writing of the manuscript.

Funding Open access funding provided by Universitá della Calabria within the CRUI-CARE Agreement.

Availability of data and material The data are available on request from the authors.

Declarations

Ethical approval No human participants or animals have been involved in this study.

Conflicts of interest/Competing interests The authors have no conflicts of interest to declare that are relevant to the content of this article.

Informed Consent Not applicable.

Code availability The implemented codes are available on request from the authors.

References

Abeysooriya RP, Bennell JA, Martinez-Sykora A (2018) Jostle heuristics for the 2D-irregular shapes bin packing problems with free rotation. Int J Prod Econ 195:12–26

Bennell J, Cabo M, Martínez-Sykora A (2018) A beam search approach to solve the convex irregular bin packing problem with guillotine cuts. Eur J Oper Res 270(1):89–102

Dargan S, Kumar M, Ayyagari M, Kumar G (2020) A survey of deep learning and its applications: a new paradigm to machine learning. Arch Comput Methods Eng 27:1071–1092

Dean HT, Tu Y, Raffensperger JF (2006) An improved method for calculating the no-fit polygon. Comput Oper Res 33(6):1521–1539

Domovic D, Rolich T, Grundler D, Bogovic S (2014) Algorithms for 2D nesting problem based on the no-fit polygon. In: 2014 37th international convention on information and communication technology, electronics and microelectronics (MIPRO), pp 1094–1099

Dong S, Wang P, Abbas K (2021) A survey on deep learning and its applications. Comput Sci Rev 40:100,379

Drake JH, Kheiri A, Özcan E, Burke EK (2020) Recent advances in selection hyper-heuristics. Eur J Oper Res 285(2):405–428

Han W, Bennell J, Zhao X, Song X (2013) Construction heuristics for two-dimensional irregular shape bin packing with guillotine constraints. Eur J Oper Res 230(3):495–504

Hu X, Li J, Cui J (2020) Greedy adaptive search: a new approach for large-scale irregular packing problems in the fabric industry. IEEE Access 8:91476–91487

Imamichi T, Yagiura M, Nagamochi H (2009) An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. Discret Optim 6:345–361

Leung Lin Z (2012) Extended local search algorithm based on nonlinear programming for two-dimensional irregular strip packing problem. Comput Oper Res 39:678–686

Liu Q, Zeng J, Zhang H, Wei L (2020) A heuristic for the two-dimensional irregular bin packing problem with limited rotations. In: Fujita H, Fournier-Viger P, Ali M, Sasaki J (eds.) Trends in artificial intelligence theory and applications. Artificial intelligence practices, pp 268–279. Springer

Lopez-Camacho E, Ochoa G, Terashima-Marín H, Burke EK (2013) An effective heuristic for the two-dimensional irregular bin packing problem. Ann Oper Res 206:241–264

Luo Q, Rao Y, Peng D (2022) Ga and gwo algorithm for the special bin packing problem encountered in field of aircraft arrangement. Appl Soft Comput 114:108,060

López-Camacho E, Terashima-Marin H, Ross P, Ochoa G (2014) A unified hyper-heuristic framework for solving bin packing problems. Expert Syst Appl 41(15):6876–6889

Martinez-Sykora A, Alvarez-Valdes R, Bennell J, Ruiz R, Tamarit J (2017) Matheuristics for the irregular bin packing problem with free rotations. Eur J Oper Res 258(2):440–455

Okano H (2002) A scanline-based algorithm for the 2D free-form bin packing problem. J Oper Res Soc Jpn 74:145–161

Pandey AA (2021) An analysis of solutions to the 2D bin packing problem and additional complexities. Int J Math Appl 9:111–118

Terashima-Marín H, Ross P, Farías-Zárate CJ, López-Camacho E, Valenzuela-Rendón M (2010) Generalized hyper-heuristics for solving 2D regular and irregular packing problems. Ann Oper Res 179(1):369–392

Wäscher GHH, Schumann H (2007) An improved typology of cutting and packing problems. Eur J Oper Res 183(1):1109–1130

Zhang H, Liu Q, Wei L, Zeng J, Leng J, Yan D (2022) An iteratively doubling local search for the two-dimensional irregular bin packing problem with limited rotations. Comput Oper Res 137:105,550