An Efficient Hyperheuristic for Strip-Packing Problems*

Ignacio Araya[1], Bertrand Neveu[1], and María-Cristina Riff[2]

[1] Project COPRIN, INRIA, Sophia-Antipolis, France
{Ignacio.Araya,Bertrand.Neveu}@sophia.inria.fr
[2] Department of Computer Science, Universidad Técnica Federico Santa María, Valparaíso, Chile
Maria-Cristina.Riff@inf.utfsm.cl

Summary. In this paper we introduce a hyperheuristic to solve hard strip packing problems. The hyperheuristic manages a sequence of greedy low-level heuristics, each element of the sequence placing a given number of objects. A low-level solution is built by placing the objects following the sequence of low-level heuristics. The hyperheuristic performs a hill-climbing algorithm on this sequence by testing different moves (adding, removing, replacing a low-level heuristic). The results we obtained are very encouraging and improve the results from the single heuristics tests. Thus, we conclude that the collaboration among heuristics is an interesting approach to solve hard strip packing problems.

Keywords: Hyperheuristic, strip packing problems, low-level heuristic, hill climbing.

1 Introduction

In this paper we focus our attention on methods to solve the two-dimensional strip packing problem, where a set of rectangles (objects) must be positioned on a container (a rectangular space area). This container has a fixed width dimension and a variable height size. The goal is, when possible, to introduce all the objects in the container without overlapping, using a minimum height dimension of the container. This problem is NP-hard and exact approaches [18, 15] are in general limited to small instances. Four variants of this problem exist, depending on the possibility of rotation of the objects, and on the presence of the guillotine cut constraint[1].

In the literature many heuristic approaches have been proposed. In our understanding the most complete review has been presented in E. Hopper's Thesis [11]. However, in the last few years the interest in this subject has increased, and so has the interest in the number of research papers presenting new approaches and improvements to the existing strategies. These approaches are in general single

* This work was partially financed by the Fondecyt Project 1060377.

[1] This constraint requires that all objects placed in the container can be reproduced by a series of guillotine cuts, i.e. edge-to-edge cuts parallel to the edges of the container.

C. Cotta et al. (Eds.): Adaptive and Multilevel Metaheuristics, SCI 136, pp. 61–76, 2008.
springerlink.com

heuristics or heuristics incorporated into metaheuristics methods. Recently, the concept of hyperheuristic has been introduced and successfully tested in different problems, [5]. The key idea is to tackle problems using various low-level heuristics and develop a framework that controls the applications of the heuristics. Using this framework the time consuming task of designing an algorithm with special components for a specific algorithm is reduced. This kind of approach is useful to obtain a good solution for a problem in a reasonable amount of time. It emphasizes a compromise between the quality of the solution and the invested time for designing the algorithm. Our goal is to show that hyperheuristics can be applied to solve Strip Packing Problems providing effective solutions in an efficient way. Our hyperheuristic is compared to other approaches using well known benchmarks. This paper is organized as follows: First we present an overview of methods based on heuristics to solve the strip packing problem, which are included in our hyperheuristic approach. Next we introduce our hyperheuristic. We will then present the results obtained using the benchmarks. Finally, our conclusions and future trends in this research area are presented.

2 Heuristics Based Methods

In this section, we present a survey of the main heuristics for strip packing problems and of the most efficient algorithms using them.

2.1 Various Low-Level Heuristics

Baker in [2] introduced Bottom-Left heuristics (BL), which first orders the objects according to their area. The objects are placed at the top and pushed down and left as much as possible. This method was improved by Chazelle [8] and called Bottom Left Fit (BLF) : each object is located at the most bottom and left possible place. Hopper [12] presented BLD which is an improved version of BL, where the objects are ordered using various criteria (height, width, perimeter, area) and the algorithm selects the best result obtained. Lesh et al. in [16] focus their research on improving BLD heuristic. They call their new heuristics BLD^*. In BLD^* the objects are randomly ordered according to the Kendall-tau distance from all of the possible fixed orders. This strategy is called Bubble Search, [17] and can be applied to any constructive algorithm in order to randomize a fixed ordering. As in GRASP, this strategy repeats greedy placements with this randomized ordering until a time limit is reached.

Another type of heuristics, Best Fit (BF) [6], uses a dynamic ordering for the rectangles to be located. The algorithm goes through the possible places from the most bottom left one, and selects for each place the rectangle that best fits in it (if it exists).

Let us now describe heuristics for problems with guillotine cut constraint. The heuristics FFDH and NFDH proposed in [14] and BFDH proposed initially in [19], and modified by [3] as BFDH* are very similar. In each of them, the objects are oriented such that their width is not lower than their height, and

they are ordered from highest to lowest. Each object is packed in a rectangular sub-area of the container in the bottom left corner. The width of the sub-area is given by the container, and the height is given by the first object packed in this sub-area. When it is possible to include the current object to be placed into some sub-areas, it is positioned into the sub-area having: the least available area for BFDH; the bottom available area for FFDH; and the top area, if it is available, for NFDH. In other cases the algorithm opens a new sub-area above the existing sub-areas positioning the current object in the bottom left corner as the first object of this sub-area. BFDH* seeks to improve this heuristic by allowing object rotations, so that when the algorithm searches to include the current object into a sub-area it tests both orientations.

Zhang et al. [21] propose a recursive heuristic HR for problems with guillotine cut constraint. When the first object is positioned in the container (on the bottom left corner) it identifies two remaining areas. It recursively continues placing the remaining objects. To improve the performance of the heuristic, the authors present a deterministic algorithm (HRalg) that gives priority to the objects with bigger areas. Zhang et al. claim that their algorithm quickly obtains good results on Hoper's benchmarks.

For our approach we have selected HR, BF, BLF, BFDH* as the low-level heuristics for problems without guillotine cut constraint, because they have shown to be individually competitive. For problems with guillotine cut constraint the selected heuristic are HR and BFDH*.

2.2 Metaheuristic Approaches

These and other low-level heuristics have been used in metaheuristic approaches, as tabu search, simulated annealing, and genetic algorithms. The first idea is to build an initial solution by a low level heuristic and to perform a local search on the layout. Neveu et al. [20] present an incremental move, which allows additions and removals of rectangles. They also implement a generic metaheuristic using this move obtaining competitive results.

Other researchers prefer to work on the order of the objects for each positioning heuristic. In [12] they present a genetic algorithm and a simulated annealing algorithm (GA+BLF and SA+BLF), both of which try to find the best order for the objects to be placed in the container using the BLF strategy.

For the case of fixed orientation problems, the best approach to our knowledge appears to be the GRASP based approach described in [1]. This approach repeats the following two-phases algorithm: the rectangles are first placed by a slightly randomized BF like constructive phase. Then the solution is improved by a strictly improving Variable Neighborhood Search (VNS).

On the other hand, Bortfeldt [3] introduced a Genetic Algorithm called SP-GAL and obtained the best results known in the literature for the problems allowing the rectangles to be rotated. The algorithm generates an initial population using a BFDH* heuristic which is an improvement of the BFDH heuristic initially proposed in [19]. This heuristic works with a layer structure, that takes into account the guillotine cut constraint. The genetic algorithm directly

performs a search in this layer structure. For problems without the guillotine cut constraint, a post-optimization procedure breaks this layer structure. The same genetic algorithm is used in [4] for bigger instances (1000 pieces). It is divided in GA-1, GA-2, GA-3 and GA-4, each of them initialized with different parameters. The procedure is only applied to problems with the guillotine cut constraint, because the post-optimization procedure is negligible for large instances [4].

Burke et al. [7] hybridize the best-fit heuristic with metaheuristic approaches such as tabu search (BF-TS), simulated annealing (BF-SA) and genetic algorithms (BF-GA). BF-SA obtains the best results.

3 The Hyperheuristic Approach: H-SP

The hyperheuristic framework manages a set of low-level heuristic and tries to find a way to apply them. There are some genetic inspired hyperheuristics in the literature to solve combinatorial problems [9,10]. However, in most of the cases, they use a representation that just corresponds to a simple sequence of low-level heuristics to be applied.

We have chosen to build a simple hyperheuristic that manages a sequence of low-level greedy heuristics. From the analysis of the four selected low-level heuristics we can remark the following:

- Performance changes according to the order of the objects and their rotation.
- The data structure to obtain a good implementation code is not always the same for all of these heuristics.

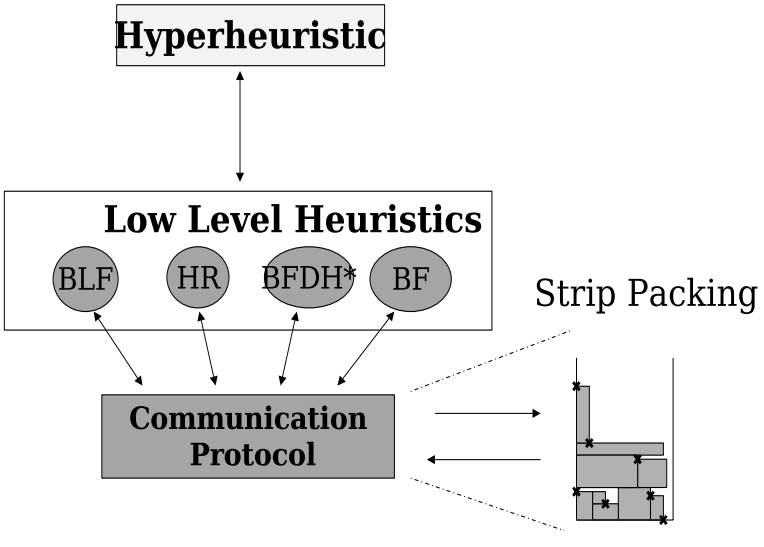

Fig. 1. H-SP: Hyperheuristic for Strip Packing

Taking into account these remarks, we have designed a new hyperheuristic approach which allows us to include a good individual implementation for each heuristic considering them as black boxes. They communicate following a protocol for modifying the current state of the search (the floor with the objects already located by the preceding heuristics and the remaining objects to locate) as shown in figure 1.

3.1 Representation

The representation structure used is a *constructive algorithm* formed by the sequential composition of constructive heuristics among a set H. A **configuration** X is thus a constructive algorithm:

$$X = h_1(p_1, n_1) * h_2(p_2, n_2) * ... * h_k(p_k, n_k) \tag{1}$$

Where $h_1, ..., h_k \in H$ are the constructive heuristics, $p_1, ..., p_k \in P$ are parameters to initialize the heuristics and n_i is an integer number that represents the amount of pieces that the heuristic h_i must place. $*$ is the sequential composition operator.

The sets P and H depend on the kind of problem that will be solved (with or without guillotine constraint, with or without rotation allowed).

Let N be the number of pieces to place inside the container. The next two constraints must be satisfied:

$$n_i > 0, \forall i = 1...k \tag{2}$$

$$\sum_{i=0}^{k} n_i = N \tag{3}$$

The parameters p_i are related to the order and the rotation of the pieces before the placement. The basic order criteria used are: decreasing heights (DP), decreasing widths (DW), decreasing areas (DA) and decreasing perimeters (DP). The rotation criteria used are: width greater or equal than the heights ($W \geq H$), heights greater or equal than the widths ($H \geq W$), rotate no object (NR) and rotate all the objects (All_R).

Figure 2 shows a configuration example with 3 heuristics. To translate the configuration into the problem, the heuristics are evaluated sequentially. The first is BLF, the parameters p indicate that the rectangles must be ordered by decreasing weights (DW) and rotated with their widths greater or equal than their heights ($W \geq H$). Just when the process of ordering and rotation has been realized, the BLF heuristic will begin to place the pieces inside the container ($n = 4$ pieces, corresponding to the white rectangles). The rectangle numbers indicate the placement order of the pieces.

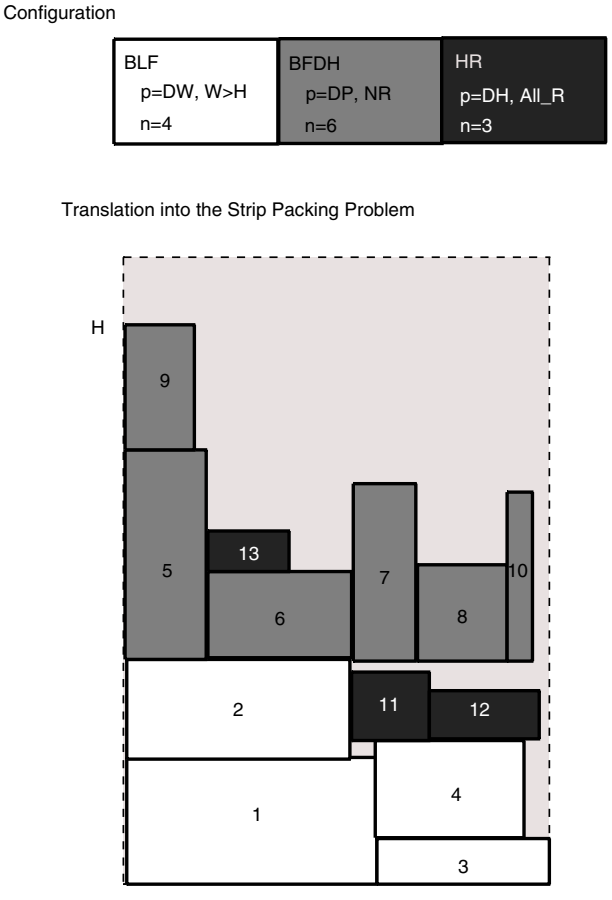

Fig. 2. Configuration example

3.2 Moves

The local search operations that we have defined in our high-level structure allow heuristics to be added, deleted and replaced from the configuration. These operations are applied with equal probability.

Let the current configuration:

$$X_C = h_1(..) * ... * h_{i-1}(p_{i-1}, n_{i-1}) * h_i(p_i, n_i) * h_{i+1}(p_{i+1}, n_{i+1}) * ... * h_k(..) \quad (4)$$

The **add operation** selects random values $i \in \{1..k\}$, $h_{add} \in H$, $p_{add} \in P$ and $n_{add} \in \{1..n_i\}$. The return of the operation is a new configuration:

$$X'_C = ... * h_{i-1}(..) * h_{add}(p_{add}, n_{add}) * h_i(p_i, n_i - n_{add})... \quad (5)$$

If $n_i - n_{add}$ is equal to 0, the heuristic h_i is simply eliminated from the configuration. The key idea of this operation is to include new heuristics in a different step of the algorithm in order to obtain a better cooperation among them.

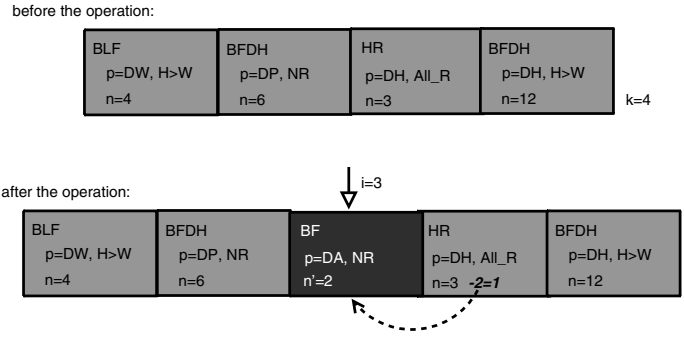

Fig. 3. Example of the *add operation*

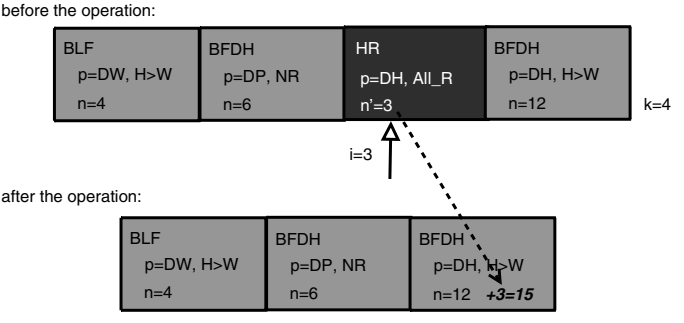

Fig. 4. Example of the *remove operation*

Figure 3 shows an example. The new heuristic is located in the third position of the configuration, reducing by $n'(2)$ the next heuristic n value.

The **remove operation** selects a random value $i \in \{1..k\}$. The return operation is a new configuration:

$$X'_C = ... * h_{i-1}(p_{i-1}, n_{i-1}) * h_{i+1}(p_{i+1}, n_{i+1} + n_i) * ... \tag{6}$$

If the random value of i is equal to k, then:

$$X'_C = ... * h_{k-1}(p_{k-1}, n_{k-1} + n_k) \tag{7}$$

The idea here is to allow the algorithm to discard some heuristics obtaining better results without them.

Figure 4 shows an example. The third heuristic is removed from the configuration and the value of $n'(3)$ is added to the next heuristic n value.

The **replace operation** selects random values $i \in \{1..k\}$, $h_{rep} \in H$ and $p_{rep} \in P$. The operation returned is a new configuration:

$$X'_C = ... * h_{i-1}(..) * h_{rep}(p_{rep}, n_i) * h_{i+1}(..) * ... \tag{8}$$

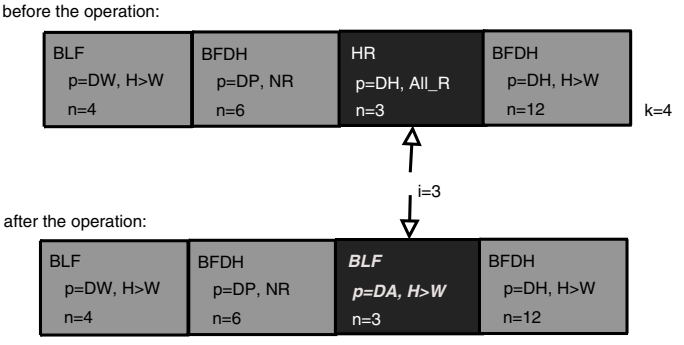

Fig. 5. Example of the *replace operation*

The idea of this operation is to give more exploration capability to the algorithm.

Figure 5 shows an example. The third heuristic in the configuration is replaced by a new one, with new parameters p and the same value of n.

All these operations maintain the constraints (2) and (3) satisfied. The representation and the defined operations allow the hyperheuristic algorithm to reach a wider combination between low-level heuristics.

3.3 Evaluation Function

Our approach uses the traditional fitness function for strip-packing [12], which is to minimize the container's height used. It is supposed that the container's width is fixed. The quality of a constructive algorithm or configuration is evaluated according to the quality of the solution that it obtains.

3.4 Procedure

The hyperheuristic explores the space of constructive algorithms (Xs) by starting from an initial and random generated configuration (X_0). To do that, our approach follows a Hill-climbing procedure, thus in each iteration it is applied one random operation to the algorithm and if the new algorithm X'_C is better or equal than the current one (X_C), then X'_C will be the new algorithm for the next iteration.

In order to escape local minima, we have performed for each H-SP test 10 restarts. It means that one execution of H-SP of 100 seconds corresponds to 10 hill-climbing procedures of 10s each.

The initial algorithm is $X_0 = h_1(p_1, n_1) * h_2(p_2, n_2) * ... * h_m(p_m, n_m)$, with $h_1 \neq h_2 \neq ... \neq h_m$ and $m = \#H$, in other words, *all* the heuristics are used once to construct X_0. The values of p_i are selected at random from the set P, and the values of n_i are fixed satisfying the equation (9):

$$n_i = \frac{i \times N}{m} - \sum_{j=1}^{i-1} n_j \tag{9}$$

Algorithm 1. H-SP($Time\_Limit$)

```
for i = 1 to 10 do
  restart_time()
  X_0 ← RandomAlgorithm(H, P, N)
  Best_Algorithm ← X_0
  X_C ← X_0

  while time()< Time_Limit/10 do
    select RandomNumberFrom(1..3)
      case 1: X'_C ← Add(X_C)
      case 2: X'_C ← Remove(X_C)
      case 3: X'_C ← Replace(X_C)
    end select

    if Evaluate(X'_C) ≥ Evaluate(X_C) then
      X_C ← X'_C
    end if
  end while

  if Evaluate(X_C) ≥ Evaluate(Best_Algorithm) then
    Best_Algorithm = X_C
  end if
  return Best_Algorithm
end for
```

For example, if $m = 4$ ($\#H$ is also 4) and the amount of pieces N is 47, the four heuristics in set H will be selected in some order, the parameters p_i will be randomly selected from P and the values of n_1, n_2, n_3 and n_4 will be respectively 11, 12, 12 and 12.

Algorithm 1 shows the procedure. *RandomAlgorithm* function generates the initial constructive algorithm. *Add*, *Remove* and *Replace* functions, perform the operations described in 3.2. *Evaluate* function executes the generated algorithms and obtains their fitness. Finally the best solution can be obtained executing the *Best_Algorithm*.

4 Tests

We have performed two kinds of tests. The first one compares the results obtained using low-level heuristics with the results of our hyperheuristic approach. We report the quality of the solution found and the percentage used of each single low-level heuristic in the hyperheuristic. The second test compares H-SP with the best reported results from the strip-packing state of the art.

4.1 Benchmarks

For these tests we use the 21 Hopper's instances classified in 7 classes $C_1, \ldots, C_7$, according to their size. The optimal solution of each instance is known, [12]. We

Table 1. Gap to the solution for low-level heuristics and H-SP

Class	**Low-level heuristics**				**H-SP**20s	
	BLF	HR	BFDH*	BF	Average	Best
C1	6.6	6.6	6.6	5	**0**	**0**
C2	13.3	8.8	8.8	8.8	0.89	**0**
C3	11.1	6.6	6.6	6.6	**2.22**	**2.22**
C4	4.4	3.8	3.8	3.3	**1.67**	**1.67**
C5	2.6	2.6	2.6	2.6	1.26	**1.11**
C6	3.1	2.7	2.7	2.5	1.28	**0.83**
C7	2.6	2.6	2.6	2.2	1.17	**0.97**
Average	6.24	4.81	4.81	4.42	1.21	**0.97**

also report the results obtained using Bortfeldt's problems that have been recently proposed in [3]. He has defined 360 instances of strip-packing problems with 1000 rectangles and unknown optimal solutions. There are 12 sets of problems and 30 instances belonging to each set. They differ in four factors related to the objects to be placed: width, area, heterogeneity and maximum dimension ratio.

The hardware platform for the experiments was a PC Pentium IV, 2.66Ghz with 1024 MB RAM under Debian operating system. The algorithm has been implemented in C++.

4.2 Comparison with Low-Level Heuristics

The Table 1 shows the results, using Hopper's instances and allowing rotation, found by each single heuristic and the average and the best results obtained by our H-SP algorithm over 10 runs. In order to compare H-SP with low-level heuristics, we limited the running time of H-SP to 20 seconds.

The set H, in this test, is composed of the heuristics BLF,HR,BFDH*,BF, in their original versions [2]. And the set of parameters P is composed of all combinations of types of ordering (7) and types of rotation (4) for the remaining objects.

Each low-level heuristic is evaluated with each parameter in P ($7 \times 4 = 28$) and the best solution is shown, the time for each instance is not superior to 1 second. The results are calculated as the percentage from the optimal solution ($gap(\%) = \frac{solution-opt}{opt}$).

The quality of the solution found by the low-level heuristics has been strongly improved by the final constructive algorithm X_F given by our framework. The execution time of X_F is comparable to the execution time of low-level heuristics

[2] Originally each heuristic can decide when rotate or not an object, for the case of no rotation allowed instances, this functionality is not used.

Table 2. Average use of low-level heuristics in H-SP

Class	**Low-level heuristics**			
	BLF	HR	BFDH*	BF
C1	9.24	30.00	54.45	6.27
C2	11.14	27.17	11.95	49.72
C3	29.71	15.80	0.23	54.24
C4	34.46	24.41	8.52	32.59
C5	40.19	10.76	3.70	45.33
C6	15.82	10.97	1.95	71.25
C7	99.68	0.31	0	0
Average	34.48	13.07	5.49	46.94

Table 3. Gap to the solution for Hopper's instances with rotation allowed (RF)

Class	GA+ BLF	SA+ BLF	HRalg	SPGAL		**H-SP**100s		**H-SP**1000s	
				Average	Best	Average	Best	Average	Best
C1	4	4	8.33	1.7	1.7	**0**	**0**	**0**	**0**
C2	7	6	4.45	0.9	**0**	**0**	**0**	**0**	**0**
C3	5	5	6.67	2.2	2.2	2.22	2.22	1.78	**1.11**
C4	3	3	2.22	1.4	**0**	1.67	1.67	1.67	1.67
C5	4	3	1.85	**0**	**0**	1.11	1.11	1.11	1.11
C6	4	3	2.5	0.7	**0.3**	1	0.83	0.83	0.83
C7	5	4	1.8	0.5	**0.3**	1.03	0.97	0.69	0.56
Average	4.57	4	3.97	1.06	**0.64**	1	0.97	0.87	0.75

(in C7 instances, X_F and BLF take $0.0045s$ and $0.0035s$, respectively, to construct a solution).

In Table 2, we report the *average percentage of pieces* that each heuristic of the set H places in the final constructive algorithm X_F for each kind of problem.

We can remark that each problem requires a different combination of the low-level heuristics. This is the advantage of the implicit natural adaptation of the hyperheuristic framework. We remark that BFDH* tends to be less applied as the size of the problem increases, while BLF shows the exact contrary behavior. A pattern cannot be identified for both BF and HR heuristics. Note however that BF has been used more frequently than HR. In addition, HR is more useful in solving smaller problem categories. Thus, the application percentage of the low-level heuristics depends on the problem instance to be solved. Furthermore, the algorithm is able to self-adapt to the problem at hand.

Figure 6 shows a typical final constructive algorithm and its solution for a class C7 instance (especifically the C72 instance).

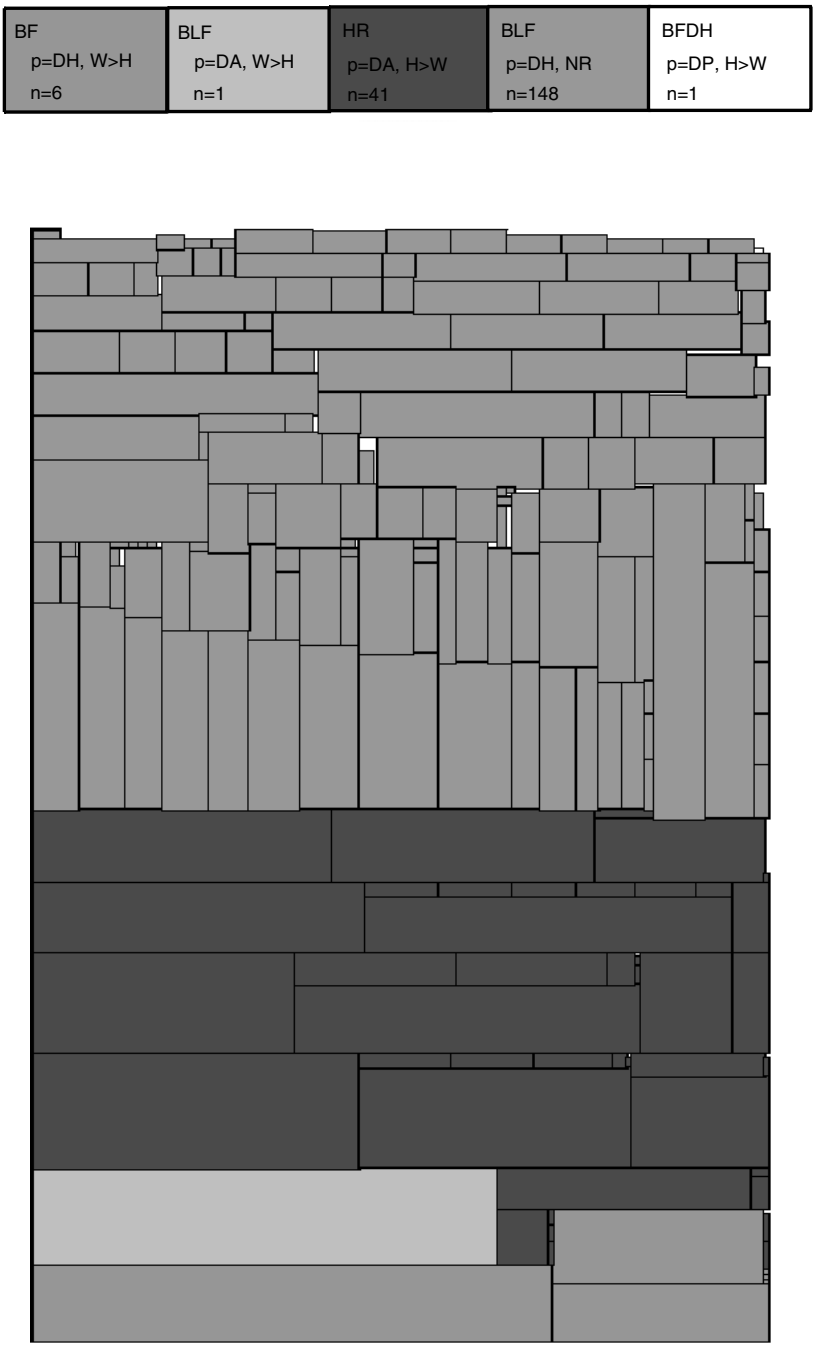

Fig. 6. Solution for instance C72

Table 4. Gap to the solution for Hopper's instances without rotation (OF)

Class	Iori algorithm	BF+ SA	SPGAL Best	GRASP Average	GRASP Best	**H-SP**100s Average	**H-SP**100s Best	**H-SP**1000s Average	**H-SP**1000s Best
C1	1.67	0	1.67	**0**	**0**	1.33	**0**	1	**0**
C2	2.22	6.25	2.22	**0**	**0**	**0**	**0**	**0**	**0**
C3	2.22	3.33	3.33	**1.11**	**1.11**	2.22	2.22	2.22	2.22
C4	4.75	1.67	2.78	**1.67**	**1.67**	2.11	**1.67**	**1.67**	**1.67**
C5	3.93	1.48	1.48	**1.11**	**1.11**	1.18	**1.11**	1.26	**1.11**
C6	4.00	1.39	1.67	1.58	1	1.39	1.11	1.22	**0.83**
C7	—	1.77	1.25	1.39	1.25	1.08	**0.97**	1	**0.97**
Average	3.13	2.27	2.06	0.98	**0.88**	1.33	1.01	1.2	0.97

4.3 Comparison with State-of-the-Art Algorithms

Tables 3 and 4 summarize the best results found in the literature [1, 3, 4, 12, 13, 16, 18, 21], and the results obtained by our hyperheuristic for the Hopper's instances. The results are calculated as the percentage from the optimal solution ($gap(\%) = \frac{solution-opt}{opt}$).

Tests with rotation allowed (RF)

We have first studied the problems where the rotation of the rectangles is allowed. Table 3 shows the results found in the literature for some algorithms compared with H-SP. The algorithms GA+BLF and SA+BLF [12], were run on a Pentium Pro 200 MHz with an average time per run of 674 minutes for SA+BLF and 136 minutes for GA+BLF. The deterministic algorithm HRalg [21], was run on a 2.4GHz CPU, with an average time per run of 5.59 seconds (0 seconds for C1 instances, 36 seconds for C7). SPGAL [4] reports an average time per run of 159 seconds on a 2GHz Pentium and the algorithm was run 10 times for each instance. The H-SP algorithm have been run 10 times with execution times of 100 and 1000 seconds for each instance. The set H is composed of the heuristics BLF, HR, BFDH* and BF, in their original versions.

Results in Table 3 show that H-SP gives good quality solutions and even better solutions than various other algorithms for the problem (metaheuristics and heuristics) except for the SPGAL algorithm. This algorithm is especially designed for these benchmarks and evaluates all possible rotations for each object to be positioned.

Tests without Rotation (OF)

We have also tested the algorithms considering the same benchmarks, but without allowing object rotations. To this test, the set H is composed of the heuristics BLF, HR, BFDH* and BF, in their no-rotation-allowed versions. The set P is reduced to only order parameters (rotation have no sense).

Table 5. Gap to the solution for Bortfeldt's instances

Set of Problems	**Type RG** GA4	H-SP 100s	**Type OG** GA4	H-SP 100s	**Type RF** H-SP 100s	H-SP 1000s
1	**2.44**	3.39	**4.43**	4.89	1.44	1.01
2	**1.86**	1.92	3.79	**3.70**	0.99	0.74
3	2.61	**1.54**	3.07	**2.32**	1.26	1.07
4	2.34	**1.04**	2.85	**1.64**	0.75	0.62
5	**1.27**	3.11	**2.08**	4.12	1.07	0.82
6	**1.04**	1.67	**1.68**	2.38	0.76	0.61
7	1.87	**1.59**	2.39	**2.13**	1.60	1.46
8	**1.18**	1.51	**1.62**	1.92	1.08	0.92
9	3.03	**2.12**	4.34	**3.45**	1.25	0.76
10	1.78	**1.27**	1.67	**1.52**	0.52	0.38
11	1.87	**1.46**	2.45	**1.97**	1.32	1.12
12	1.83	**1.58**	2.12	**2.03**	0.61	0.54
Average	1.93	**1.85**	2.71	**2.67**	1.05	0.84

Table 4 shows the results found by some algorithms compared with H-SP. The GRASP algorithm has been run 10 times on a 2GHz Pentium, the stopping criterion is of 60 seconds. BF+SA [7] has been run 10 times on a 2GHz Pentium with a limit of 60 seconds per run. Iori et al. [13] algorithm was run 300 seconds on a Pentium III at 800Mhz. SPGAL [4] reports an average time per run of 160 seconds on a 2GHz Pentium and the algorithm was run 10 times for each instance. The H-SP algorithm have been run 10 times with execution times of 100 and 1000 seconds for each instance.

Up to now, GRASP was the best approach. We obtained better average results than GRASP in the two biggest classes (C6 and C7).

4.4 Tests with Bortfeldt's Instances

We performed three series of tests with the 360 large new random instances proposed by Bortfeldt and Gehring [4], subdivided in 12 sets of 30 instances. For all these instances, the optimal solution is not known. We use as performance index the gap with the continuous lower bound clb [4] ($gap(\%) = \frac{(best_{found} - clb)}{clb}$).

In Table 5 we have compared the Bortfeldt's algorithm GA4 (based on SP-GAL) with H-SP. In the second and third columns we consider the problems type **RG**, that requires guillotine cuttings and allows objects to be rotated. For these set of problems the average execution time of algorithm GA4 is 895 seconds on a 2GHz Pentium. For these guillotinable instances, the set H is composed of low-level heuristics that respect that guillotine constraint. The heuristics are only two: HR and BFDH* (Section 2.1). For each problem instance the hyperheuristic is run once with a maximum execution time of 100 seconds.

In the fourth and fifth columns we consider the problems type **OG**, that requires guillotine cuttings and where the orientation of the objects is fixed. We used the same low-level heuristics as for RG instances. The average execution time for Bortfeldt's algorithm is 717 seconds on a 2GHz Pentium. For each problem instance the hyperheuristic is run once with a maximum execution time of 100 seconds and the average results are shown.

We also considered the problems type **RF** shown in the last column, where guillotine cutting is not required and the objects may be rotated. The set H is composed of the heuristics: BLF, HR, BFDH* and BF, in their original versions. For each problem instance the hyperheuristic is run once with maximum execution times of 100 and 1000 seconds.

We can remark that we are competitive for all these RG and OG benchmarks with Bortfeldt's algorithm. Moreover with the type RF we can see that we reduced the gap obtained for the RG and OG problems. This behavior was expected, since RF problems are less constrained, nevertheless, Bortfeldt and Gehring say that their algorithms (GA-4 is the best of them) obtain negligible improvements when they are applied with the post-optimization process [4], in other words, when they are applied to RF problems.

Our framework is flexible: we only had to change the set of low-level heuristics in each case, and the framework gives us competitive results.

5 Conclusions

This research allows us to conclude that using a hyperheuristic approach can improve the performance of single greedy heuristics. Moreover, the hyperheuristic is able to adapt itself to the problem by selecting a good combination of these low-level heuristics. This framework is quite general: we have shown that it could solve different strip packing problems (RF, OF, RG, OG). For solving a new problem type, the major task is the selection of suitable and efficient low-level heuristics. The hyperheuristic framework will allow cooperation among them, hopefully improving their single behaviors.

For future works, we believe that adding new operations and low-level heuristics can obtain configurations that explore in a better way the search space.

Acknowledgments

This work was carried out in the context of the Chile-France INRIA/CONICYT collaboration project.

References

1. Alvarez-Valdes, R., Parreño, F., Tamarit, J.M.: Reactive grasp for the strip packing problem. In: Proceedings Metaheuristic Conference MIC (2005)
2. Baker, B.S., Coffman, E.G., Rivest, R.L.: Orthogonal packings in two dimensions. SIAM Journal on Computing 9, 846–855 (1980)

3. Bortfeldt, A.: A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. European Journal of Operational Research 172, 814–837 (2006)
4. Bortfeldt, A., Gehring, H.: New large benchmarks for the two-dimensional strip packing problem with rectangular pieces. In: IEEE Proceedings of the 39^{th} Hawaii International Conference on Systems Sciences (2006)
5. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: an emerging direction in modern search technology. Handbook of Meta-heuristics 16, 457–474 (2003)
6. Burke, E., Kendall, G., Whitwell, G.: A new placement heuristic for the orthogonal stock-cutting problem. Operations Research 52, 697–707 (2004)
7. Burke, E., Kendall, G., Whitwell, G.: Metaheuristic enhancements of the best-fit heuristic for the orthogonal stock cutting problem. Technical report, Univ. of Nottingham, Computer Science Technical Report No. NOTTCS-TR-2006-3 (2006)
8. Chazelle, B.: The bottom left bin packing heuristic: an efficient implementation. IEEE Transactions on Computers 32, 697–707 (1983)
9. Cowling, P., Kendall, G., Han, L.: An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In: Congress on Evolutionary Computation, CEC 2002, pp. 1185–1190 (2002)
10. Cowling, P., Kendall, G., Soubeiga, E.: Hyperheuristics: A robust optimisation method applied to nurse scheduling. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 7–11. Springer, Heidelberg (2002)
11. Hopper, E.: Two-Dimensional Packing Utilising Evolutionary Algorithms and other Meta-Heuristic Methods. PhD. Thesis Cardiff University (2000)
12. Hopper, E., Turton, B.C.H.: An empirical investigation on metaheuristic and heuristic algorithms for a 2d packing problem. European Journal of Operational Research 128, 34–57 (2001)
13. Iori, M., Martello, S., Monaci, M.: Metaheuristic algorithms for the strip packing problem, pp. 159–179. Kluwer Academic Publishers, Dordrecht (2003)
14. Coffmann Jr., E., Garey, D., Tarjan, R.: Performance bounds for level oriented two-dimensional packing algorithms. SIAM Journal on Computing 9(1), 808–826 (1980)
15. Lesh, N., Marks, J., Mahon, A.Mc., Mitzenmacher, M.: Exhaustive approaches to 2d rectangular perfect packings. Information Processing Letters 90, 7–14 (2004)
16. Lesh, N., Marks, J., Mahon, A.M., Mitzenmacher, M.: New heuristic and interactive approaches to 2d rectangular strip packing. ACM Journal of Experimental Algorithmics 10, 1–18 (2005)
17. Lesh, N., Mitzenmacher, M.: Bubble search: A simple heuristic for improving priority-based greedy algorithms. Information Processing Letters 97, 161–169 (2006)
18. Martello, S., Monaci, M., Vigo, D.: An exact approach to the strip-packing problem. INFORMS Journal of Computing 15, 310–319 (2003)
19. Mumford-Valenzuela, C., Vick, J., Wang, P.Y.: Heuristics for large strip packing problems with guillotine patterns:An empirical study, pp. 501–522. Kluwer Academic Publishers, Dordrecht (2003)
20. Neveu, B., Trombettoni, G., Araya, I.: Incremental move for strip-packing. In: Proceedings of ICTAI 2007, Patras, Greece (2007)
21. Zhang, D., Kang, Y., Deng, A.: A new heuristic recursive algorithm for the strip rectangular packing problem. Computers and Operations Research 33, 2209–2217 (2006)