An Evolutionary Hyperheuristic to Solve Strip-Packing Problems*

Pablo Garrido and María-Cristina Riff

Department of Computer Science, Universidad Técnica Federico Santa María,
Valparaíso, Chile,
e-mail:{pgarrido, mcriff}@inf.utfsm.cl

Abstract In this paper we introduce an evolutionary hyperheuristic approach to solve difficult strip packing problems. We have designed a genetic based hyperheuristic using the most recently proposed low-level heuristics in the literature. Two versions for tuning parameters have also been evaluated. The results obtained are very encouraging showing that our approach outperforms the single heuristics and others well-known techniques.

Keywords: Hyperheuristic, Strip Packing, Evolutionary Algorithms.

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. In the literature many approaches have been proposed. In our understanding a more complete revision has been presented in E. Hopper's Thesis [10]. However, in the last few years the interest in this subject has increased, as has the interest in the number of research papers presenting new approaches and improvements to the existing strategies. These approaches are in general single heuristics or heuristics incorporated into metaheuristics methods. Recently, the concept of hyperheuristic has been introduced and tested successfully 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 emphasises a trade-off between the quality of the solution and the invested time for designing the algorithm.

Our goal in this research is to show that our hyperheuristic can be applied to solve difficult Strip Packing Problems giving good quality solutions in an

* Partially Supported by the Fondecyt Project 1060377

efficient way from both points of view: running and designing time. Our approach is compared using well known benchmarks. This paper is organised 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 framework. 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 briefly revision of the most recently published heuristics for strip packing problems that are relevant for our approach. Baker introduced [2] Bottom-Left (BL) heuristics, which orders the objects according to their area. The objects are then located on the most bottom and left coordinates possible. BL has been improved by Chazelle [7] using Bottom-Left-Fill (BLF) algorithm. Hopper [11] presented BLD which is an improved strategy 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 [14] concentrate their research to improve BLD heuristic. They proposed BLD^* where the objects are randomly ordered according to the Kendall-tau distance from all of the possible fixed orders. This strategy is called Bubble Search, [14], the key of the algorithm is the order of the objects to be placed and the object rotation capability. The results reported indicate that the top-right corner is the most suitable decision, and the most effective order is from their minimal length. Bortfeldt [3] introduced a Genetic Algorithm called SPGAL and claimed that it obtained the best results known in the literature. The algorithm generates an initial population using a $BFDH^*$ heuristic which is an improvement on the BFDH heuristic initially proposed in [16]. This heuristic works as follows: The objects are oriented such that their width is no 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. In some cases when it is possible to include the current object to be placed on some sub-areas, it is positioned in the sub-area having the least available area. 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. As we mentioned before $BFDH^*$ seeks to improve this heuristic by doing the following: It allows object rotations, so that when the algorithm searches to include the current object into a sub-area it tests both orientations and selects the best. Prior to create a new sub-area the algorithm searches the holes produced to the right of the sub-areas, dividing the available holes on guillotinable holes. It then tries to include the bigger object in the hole farthest left of the available area. Burke et al. [6] proposed the constructive algorithm Best-Fit (BF). The order in which the rectangles are placed into the strip depends on the layout of the partial solution, and the rectangle fitting best into

this layout is selected. Zhang et al. [19] propose the heuristic HR, introducing a recursive algorithm which locates the objects on the bottom left corner. When the first object is positioned in the container it identifies the two remaining areas. It recursively continues placing objects from the biggest area to the lowest area. The algorithm gives priority to the objects with bigger areas. The authors claim that their algorithm quickly obtains the best results on Hopper's benchmarks.

It seems that the key idea is to find a good order of the objects for any positioning heuristic. In [18] they present a genetic algorithm and a simulated annealing algorithm, both of which try to find the best order for the objects to be placed in the container using the BLF strategy. For our hyperheuristics we have selected HR, BF, BLF, $BFDH^*$ as the low-level heuristics, because they are shown to be individually competitive. However, some small adaptations are required for the heuristics designed for guillotinable problems.

3 The evolutionary hyperheuristic approach

From the analysis of the four low-level heuristics we can remark the following:

- Performance changes according to the order of the list of the objects, their rotation, and their location (i.e. right or left on the floor).
- The data structure to obtain a good implementation code is not always the same for all of these heuristics.

Taking into account these remarks we have designed an evolutionary 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 both interchanging and cooperation of the current state of the search. Our representation includes the following components: Heuristic H, Number of objects to be placed using H, n_H. The type of ordering of the list of the n_H objects assigned, and finally if H must consider the objects rotated or no.

In this paper we are interested in evaluating a genetic based hyperheuristic which is able to use population capabilities to combine the different heuristics according to each individual fitness. In the following section we describe this approach called G-SP.

3.1 The Genetic Inspired Hyperheuristic: G-SP

Here we propose a new hyperheuristic that is based on genetic algorithms. There exist some genetic inspired hyperheuristics in the literature to solve combinatorial problems, [8], [9]. However, in most of the cases they use a representation that just corresponds to a simple sequence of low-level heuristics to be applied.

Representation In our approach, we have defined a representation that is able to manage and to exploit more information. We have divided the low-level heuristics according to their functionality. Thus, we distinguish among greedy, ordering

and rotation heuristics. This kind of representation allows the algorithm to have a wider combination between low-level heuristics. The chromosome has also included the number of objects to be positioned using each low-level heuristics combination. The chromosome structure is shown in figure 1. In this chromosome we can identify that the algorithm must use the first low-level heuristic using the second ordering heuristic applying the fourth rotation heuristic to locate the first five objects. Note that the chromosome has not a fixed size.

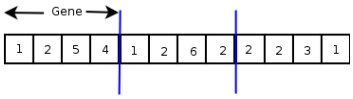

Figure1. Chromosome Structure

Specialised Genetic Operators The algorithm has four operators. One recombination operator and three mutation-like operators.

- Recombination Operator named Cross-OP: In our approach the recombination operator is an one-point crossover. The cross-point is selected such that a cut inside on the gen single structure is forbidden. It takes two parents to generate two offsprings. After crossing, the operator must do a post-procedure in order to respect the number of objects to be placed by each individual. Either the lacks or the excess of the number of objects are distributed evenly among the genes in the chromosome structure. The goal of this operator, in our approach, is to do exploration of the search space of the low-level heuristics.
- Asexual Operators: Each operator has an especial rôle.
  - Add-OP: The algorithm randomly selects a heuristic H_s from the representation of a selected chromosome. This heuristic has N_s objects to be positioned. A new heuristic is included after H_s. The new heuristic is required to position $n_1 \leq N_s$ of the objects previously assigned to H_s. The n_1 value is randomly selected. The key idea of this operation is to include new heuristics in a different step of the algorithm in order to obtain better cooperation among them.
  - Delete-OP: The algorithm randomly chooses a heuristic from the selected chromosome. The heuristic is then deleted and the number of objects previously assigned, to be located by it, are added to the objects of the previous heuristic. Thus, the algorithm is able to discard some heuristics that are not being relevant to improve its performance.
  - Replace-OP: The algorithm randomly selects both a heuristic to be replaced and the heuristic to be included. The new heuristic included inherits the number of objects to be placed. The other components of its representation are randomly generated. The idea of this operation is to give more exploration capability to the algorithm.

Evaluation Function and Selection Our approach uses the traditional fitness function for strip-packing [11], that is to minimise the container's height used. It is supposed that the container's width is fixed. A minimisation Roulette Wheel selection is implemented in order to increase the probability of choosing an individual with low height values.

Hyperheuristic Algorithm Figure 2 shows the hyperheuristic structure. The procedure create_population randomly generate the initial population of individuals. The evaluate_sequence_heuristics applies the low-level heuristics in the order that they appear on the chromosome and it evaluates the chromosome.

```
Pseudocode of G-SP
Begin
iter=0
create_population(pop_size, chromosome_init_size, gene_type)
while max_iter < iter do
   individuals = get_population()
   evaluate_heuristics_sequences(individuals)
   update_individuals_fitness()
   if random < cross_probability
   Cross-OP();
   if random < delete_probability
   Delete-OP();
   if random < add_probability
   Add-OP();
   if random < replace_probability
   Replace-OP();
   update_population()
   iter++
end while
End
```

Figure2. Structure of the Hyperheuristic G-SP

3.2 Tuning

The performance of an evolutionary algorithm strongly depends on its parameter values. Because our hyperheuristic is based on an evolutionary approach it experiments the same sensitivity problem. Given that, we have evaluated two approaches to estimate good parameter values for our hyperheuristic. The first one, called ST (i.e., standard tuning), is the classical generate and test procedure. The second approach, called RV, is a new one based on REVAC approach, recently proposed in [17] which uses statistical properties. In the test section we evaluate both schemas.

Tuning using REVAC. The Relevance Estimation and Value Calibration Approach (REVAC) [17] has been used for tuning. Roughly speaking, REVAC is a

genetic algorithm that uses some statistical properties to determine the better parameter values and also to discard some genetic operators that, with a statistical significance, do not really improve the algorithm to be tuned. It is based on the shannon entropy to measure solutions diversity. The method has shown to be effective, but is a time consuming task because it finds the better values by evaluating many runs of all the problem instances. For this, we have selected the hardest instances that really seem to require the investment in this additional computational effort. The operator's probabilities obtained can be seen in the table 1.

	ST	RV
Cross-OP	0.3	0.346
Delete-OP	0.33	0.720
Add-OP	0.33	0.323
Replace-OP	0.33	0.282

Table1. Operator's probabilities tuned using Standard Tuning and REVAC estimations

As can we observe, the Standard Tuning (ST) version indicates that the Cross-OP probability is quite less employed than the mutation operators and each asexual operator can be applied with equal probability. For REVAC (RV) we have selected the six hardest instances from the 21 problem instances. The running time for each instance has been fixed in 30 seconds (3 minutes for each instance set). The number of iterations done by REVAC, as it has been recommended by the authors, was 1000 iterations. Thus, the calibration required around 48 hours CPU time. According to the results shown in table 1, we can conclude that the four operators are significant for our hyperheuristic. Note that the higher probability value is for the operator Delete-OP.

4 Tests

We have done two kind of tests. The first one is to compare the results obtained using single low-level heuristics with our hyperheuristic approaches. We report the quality of the solution found and the percentage of each single low-level heuristic used by the hyperheuristic. The second test compares G-SP versions with the better reported results from the state of the art for strip-packing. Both tests use as benchmarks the Hopper's instances [11] for problems $C_1, \ldots, C_7$.
The hardware platform for the experiments was a PC Pentium IV Dual Core, 3.4Ghz with 512 MB RAM under the Mandriva 2006 operating system. The algorithm has been implemented in C++.

4.1 Comparison with low-level heuristics

In order to obtain significant results, the hyperheuristic has been executed 10 times for each problem category with various initial populations. We limit the running time to 60 seconds for each problem category.

Gap to the solution: The table 2 shows the percentage from both the optimal solution to the best solution found (gap $\% = \frac{(best_{found} - opt)}{opt}$) and the average for each single heuristic and for the hyperheuristic G-SP with ST and RV tuning. The quality of the solution found by each single heuristic has been strongly improved using our approaches. Furthermore, the hyperheuristic allows both a division of the task and a cooperation among the heuristics for positioning the objects.

	BLF	HR	$BFDH^*$	BF	G-SP ST	G-SP RV
C1	6.6	6.6	6.6	5	0.0	0.0
C2	13.3	8.8	8.8	8.8	4.00	4.00
C3	11.1	6.6	6.6	6.6	3.56	3.33
C4	4.4	3.8	3.8	3.3	1.78	1.67
C5	2.6	2.6	2.6	2.6	1.33	1.22
C6	3.1	2.7	2.7	2.5	1.53	1.56
C7	2.6	2.6	2.6	2.2	1.49	1.65
Average	6.24	4.81	4.81	4.42	1.95	1.91

Table2. Gap to the solution for: low-level heuristics, hyperheuristic G-SP ST and G-SP RV

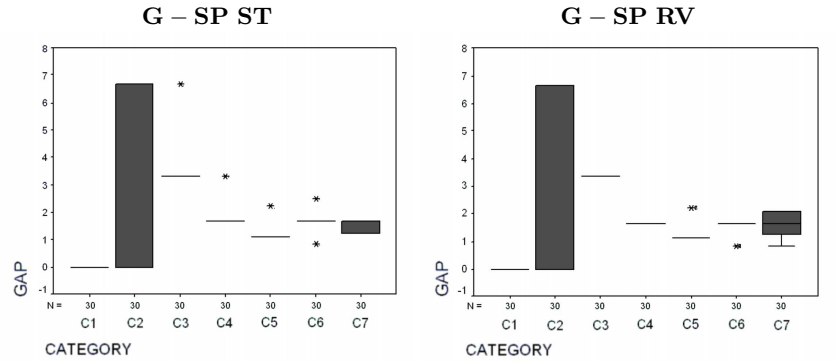

Figure3. Boxplots for both G-SP versions

Statistical Comparison: As we mentioned before, our aim is to show that a collaborative schema among simple low-level heuristics improves their individual behaviour. As the low level heuristics considered in this work are deterministic, the gap obtained by applying them at isolated is always the same. However, this is not the case in our hyperheuristics because they could obtain different gaps for the same problem at different runs. Figure 3 shows the boxplots for both hyperheuristics for each problem category. We can observe that the biggest difference among the gaps is obtained in the category C2 for both algorithms. That is because problem 2 in category C2 is very hard for all low-level heuristics as well as for our hyperheuristics. In addition, the hyperheuristic G-SP RV shows more stability than G-SP ST. The above is especially remarkable for problems on categories C3, C4 and C6.

Low-level heuristics runs: In table 3 we report the percentage of the number of times that each heuristic has been applied for each type of problem in our best genetic based hyperheuristic approach for the best heuristics combination.

	C1	C2	C3	C4	C5	C6	C7	Avg.
BLF	45.43	41.60	50.12	54.49	51.55	58.21	66.91	52.62
HR	15.65	9.87	7.02	1.43	3.93	2.47	4.72	6.44
BFDH*	3.50	28.80	2.74	0.88	1.19	0.55	0.27	5.42
BF	35.42	19.73	40.12	43.20	43.33	38.76	28.10	35.52

Table3. Average use of low-level heuristics in the G-SP RV version

This table can be interpreted as the number of the objects (in percentage) that each heuristic located on the floor. We can appreciate 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. A more detailed comparison of the use of the low-level heuristics is shown in figure 4. The figures show that BFDH* tends to be less applied as the size of the problem increases. On the contrary, BLF shows exactly the opposite behaviour. A pattern can not 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 to solve 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.

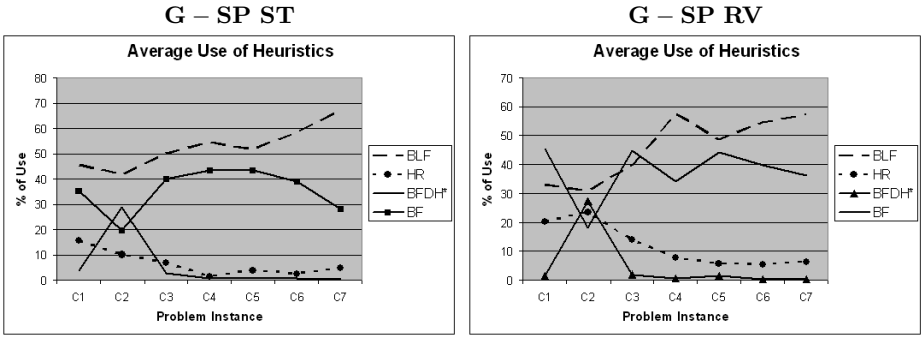

Figure4. Percentage of low-level heuristics used for G-SP

4.2 Comparison with state-of-the-art algorithms

The table 4 summarises the better results found in the literature [11], [12], [19], [3,4], [13], [1], [15], along with the results obtained by our approach for the Hopper's instances. Results show that our hyperheuristic versions obtain good quality solutions and even better than various especially-designed algorithms

	Category							
Technique	C1	C2	C3	C4	C5	C6	C7	Avg.
GA + BLF, [11]	4	7	5	3	4	4	5	4.57
SA + BLF, [11]	4	6	5	3	3	3	4	4
Iori, [12]	1.59	2.08	2.15	4.75	3.92	4.00	-	3.98
HR, [19]	8.33	4.45	6.67	2.22	1.85	2.5	1.8	3.97
SPGAL-R, [4]	1.7	0.0	2.2	0.0	0.0	0.3	0.3	0.6
SPGAL, [3]	1.59	2.08	3.16	2.70	1.46	1.64	1.23	1.98
BLD*, [13]	-	-	-	-	2	2.4	-	2.2
R-GRASP, [1]	0	0	1.08	1.64	1.10	0.83	1.23	0.84
Martello B&B, [15]	0	0	2.15	-	-	-	-	0.71
G-SP ST	0	4.00	3.56	1.78	1.33	1.53	1.49	1.95
G-SP RV	0	4.00	3.33	1.67	1.22	1.56	1.65	1.91

Table4. Gap to the solution for: state-of-the-art algorithms and G-SP tuned versions

(metaheuristics and heuristics) except for the SPGAL-R and R-GRASP algorithms that present the better solutions. These algorithms have been especially designed for these benchmarks. The above demonstrate that our approach is very competitive. In order to obtain quite good solutions, the parameters must be tuned according to the problem at hand, but this is not the main goal of hyperheuristics. For this reason, we did not invest a large amount of time tuning parameters. Instead of that, we tried to use cheap techniques to adjust the parameters to solve each problem and still giving good quality solutions. In addition, note that the values for HR in this section are not the same of the previous section. In the previous test we have fixed the running time in 60 seconds. Here the results are the best reported for this technique without imposing any time constraint.

5 Conclusions

Our research allows us to conclude that using our evolutionary hyperheuristic approach we can improve the performance of the single heuristics and some of the results obtained in the literature. That indicates that our approach is very promising to solve difficult strip packing problems. The above tell us that the hyperheuristics are strongly rich by having the following characteristics: flexible, cheap and easy to be implemented and, at the same time, are able to obtain quite good solutions. Moreover, our hyperheuristic is able to adapt itself to the problem by selecting the best combination of the low-level heuristics.
We remark that the selection of suitable low-level heuristics is a main task when we design hyperheuristics. In order to obtain competitive solutions with regard to the state of the art, we strongly require to select efficient low-level heuristics. The key idea is to allow the cooperation among them in order to improve their single behaviours.

References

1. R. Alvarez, F. Parreño, and J.M. Tamarit. Reactive grasp for the strip packing problem. *Proceedings of the 6th Metaheuristics International Conference*, 1, 2005.
2. B.S. Baker, E.G. Coffman, and R.L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9:846–855, 1980.
3. A. Bortfeldt. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, 172:814–837, 2006.
4. A. Bortfeldt and H. Gehring. New large benchmark instances for the two-dimensional strip packing problem with rectangular pieces. *IEEE Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, 2:30.2, 2006.
5. E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: an emerging direction in modern search technology. *Handbook of Meta-heuristics*, 16:457–474, 2003.
6. E. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the ortoghonal stock-cutting problem. *Operations Research*, 52:655–671, 2004.
7. B. Chazelle. The bottom-left bin packing heuristic: an efficient implementation. *IEEE Transactions on Computers*, 32:697–707, 1983.
8. P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. *Proceedings of Congress on Evolutionary Computation*, pages 1185–1190, 2002.
9. L. Han and G. Kendall. Guided operators for a hyper-heuristic genetic algorithm. *Proceedings of AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference on Artificial Intelligence*, 2903:807–820, 2003.
10. E. Hopper. *Two-Dimensional Packing Utilising Evolutionary Algorithms and other Meta-Heuristic Methods*. PhD. Thesis Cardiff University, UK, 2000.
11. E. Hopper and B.C.H. Turton. An empirical investigation on metaheuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 128:34–57, 2001.
12. M. Iori, S. Martello, and M. Monaci. *Metaheuristic algorithms for the strip packing problem*, pages 159–179. Kluwer Academic Publishers, 2003.
13. N. Lesh, J. Marks, A. Mc. Mahon, and M. Mitzenmacher. Exhaustive approaches to 2d rectangular perfect packings. *Information Processing Letters*, 90:7–14, 2004.
14. N. Lesh and M. Mitzenmacher. Bubble search: A simple heuristic for improving priority-based greedy algorithms. *Information Processing Letters*, 97:161–169, 2006.
15. S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS Journal of Computing*, 15:310–319, 2003.
16. C. Mumford-Valenzuela, J. Vick, and P.Y. Wang. *Heuristics for large strip packing problems with guillotine patterns: An empirical study*, pages 501–522. Kluwer Academic Publishers, 2004.
17. V. Nannen and A.E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. *International Joint Conference on Artificial Intelligence*, pages 975–980, 2007.
18. A. Soke and Z. Bingul. Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problems. *Engineering Applications of Artificial Intelligence*, 19:557–567, 2006.
19. D. Zhang, Y. Kang, and A. Deng. A new heuristic recursive algorithm for the strip rectangular packing problem. *Computers and Operations Research*, 33:2209–2217, 2006.