External Memory in a Hybrid Ant Colony System for a 2D Strip Packing

Carolina Salto[1], Guillermo Leguizamón[2], and Enrique Alba[3]

[1] LISI - Universidad Nacional de La Pampa, Calle 110 esq. 9, Gral Pico, La Pampa, Argentina
saltoc@ing.unlpam.edu.ar
[2] LIDIC - Universidad Nacional de San Luis, Ejército de los Andes 950, San Luis, Argentina.
legui@unsl.edu.ar
[3] GISUM - Universidad de Málaga, Campus de Teatinos, 29071, Málaga, Spain.
eat@lcc.uma.es

Abstract. In this paper we present a study of an Ant Colony System (ACS) for the two-dimensional strip packing problem. In our computational study, we emphasize the influence of incorporating an external memory, which store partial packing patterns, regarding solution quality and execution times. The stored partial solutions are used by the ants in the construction of their solutions to provide further exploitation around potential solutions. We show that our external memory based ACS algorithm to the 2SPP was able to devise solutions of quality comparable to that of those reported by an existing ACS but exhibiting low execution times.

1 Introduction

Ant Colony System (ACS) [6, 5] is one of the most representative algorithms derived from the Ant Colony Optimization (ACO) metaheuristic to deal with combinatorial optimization and other problems. It uses a colony of artificial ants which stochastically build new solutions using a combination of heuristic information and artificial pheromone trail. This pheromone trail is reinforced according to the quality of the solutions built by the ants.

Recently the idea of knowledge incorporation from previous iterations of the ACO became attractive, mainly inspired from studies in genetic algorithms. This incorporation can be made using internal or external memory [7, 13]. Under the ACO metaheuristic, the first internally implemented memory-based approach is the work of Montgomery and Randall [11], while the work of Guntsch et al.[8] is the first example of an external memory implementation. Recently, Acan [1] proposed another external memory strategies for ACO. The memory is composed of variable size solution segments obtained from elite ants from previous iterations. A segment is extracted from a solution choosing the initial and final position in a random way. Moreover the objective value of the selected solution is associated to the segment as a quality value. The solution construction process followed by an ant is modified in order to consider the information from the external memory. Based on the quality value associated to each segment, an ant takes a segment from the memory. The end component of the segment is considered by the ant as the starting point and completes the solution in the traditional way.

A posterior work from the same author [2] proposed another ACO with external memory composed of partial permutation sequences. The partial permutation sequence is obtained by randomly selecting an arbitrary number of solution components from the solution. The proposed approaches achieved significantly better solutions than conventional ACO algorithms.

In this work, we introduce an external memory based ACS to solve a two-dimensional strip packing problem (2SPP), inspired in the ideas of Acan's works. In this case, we use the external memory to store partial permutation solutions. But instead of selecting randomly positioned variable-size segments from a solution, we consider some specific knowledge from the problem, such as information about the layout of the pieces, in order to select groups of pieces to copy to the external memory. We select group of pieces that appear to lead to better solutions. In a way we follow the idea of buildings blocks of genetic algorithms. One motivation for introducing external memory is to keep track of good combination of pieces which leads levels with minimal waste. Hence, the method we propose is to cop these good levels to a new solution (exploitation of previos good information) and let the ant build the rest of the levels using the pheromone trail and the heuristic information to promote a exploration phase. The main goal of this paper is to find an improved ACS to solve a strip packing problem, and to quantify the effects of including an external memory in terms of both solution quality and the convergence speed.

The article is organized as follows. Section 2 contains an explanation of the 2SPP. Section 3 describes the hybrid ACS used to solve the 2SPP, meanwhile Section 4 presents the details of the actual external memory implementation. In Section 5, we explain the parameter settings of the algorithms used in the experimentation. Section 6 reports on the algorithm performances, and finally, in Section 7 we give some conclusions and analyze future lines of research.

2 The 2D Strip Packing Problem

Packing problems involve the construction of an arrangement of pieces that minimize the total space required for that arrangement. In this paper, we specifically consider the two-dimensional Strip Packing Problem (2SPP), which consists of a set of M rectangular pieces, each one defined by a width $w_i \leq W$ and a height h_i, $(i = 1...M)$. The goal is to pack the pieces in a larger rectangle, the *strip*, with a fixed width W and unlimited length, minimizing the required strip length; an important restriction is that the pieces have to be packed with their sides parallel to the sides of the strip, without overlapping.

In the present study some additional constrains are imposed: pieces must not be rotated and they have to be packed into three-stage level packing patterns. In these patterns, pieces are packed by horizontal levels (parallel to the bottom of the strip). Inside each level, pieces are packed bottom left justified and, when there is enough room in the level, pieces with the same width are stacked one above the other. Three-stage level patterns are used in many real applications in the glass, wood, and metal industries, and this is the reason for incorporating this restriction in the problem. The 2SPP is representative of a wide class of combinatorial problems, being a NP-hard [9] one.

Algorithm 1 ACS

```
Initialize pheromone values
repeat
  for (each ant $k$ $\{k = 1, ..., a\}$) do
    repeat
      Choose a random piece $i$ from the non selected pieces
      Open an empty level $l$ ($l.height = h_i$)
      repeat
        $J = \{1, ..., j\}$ {set of pieces that still fit in $l$}
        for ($w$ =1 to $|J|$) do
          $T$ = sum of the pheromone between $J[w]$ and all pieces $i$ already
              in the level $l$
          Divide $T$ by the number of pieces in $l$ ($|l|$)
        end for
        Choose a piece $i$ from $J$ according to the calculated probabilities
        $J = J - \{i\}$
        Update the pheromone trail {local update}
      until none of the remaining pieces fit in the level
    until all pieces are placed
    Calculate the objective value for that solution
  end for
  Apply local search to $a/2$ solutions
  Find the iteration best ant
  Replace the globally best ant if the iteration best was fitter
  Actualize the pheromone trail {global update}
until the maximum number of iterations is reached
```

3 Applying the ACS to the 2SPP

In this section we present how the ACS was adapted to solve the 2SPP (see Algorithm 1 for the general structure). The ACS description includes the heuristic information considered, pheromone trail definition, state transition rule,objective function which is calculated considering the layout obtained from a permutation of pieces, and the local search procedure used in order to improve the solution quality.

3.1 Heuristic definition

For many optimization problems, problem-dependent heuristic information can be used to give the ants additional hints for their decisions. For the 2SPP, the heuristic value for a piece j is $\eta_j = h_j$, i.e., the height of piece j.

3.2 Pheromone definition

The definition of the meaning of the pheromone trail is important to the quality of an ACS. The pheromone information should reflect the most relevant information for the process of solution construction. We maintain solutions in the form of permutations of the set of pieces [4], which will be directly translated into the corresponding packing pattern by a layout algorithm. The idea here is to favor the piece/level pair because there is a correspondence between the piece and the level where the corresponding piece will be allocated into. Trail τ_{ij} thus encodes the desirability of having a piece i and j in the same level [10]. Building blocks consisting of a partial packing in a solution (subset of several pieces to be allocated together) can emerge by reinforcing contiguous positions of the piece subset. The pheromone matrix has M rows and M columns (in a first stage, each piece is assigned to a different level, in that way initially we have M different levels).

Once all ants have completed their packing patterns, the pheromone is updated, i.e., a global pheromone updating rule is applied. In this case, only the best ant (which the respective solution is s^{best}) is allowed to place pheromone after each iteration. This is done according to the following expression:

$$\tau_{ij} = \rho \times \tau_{ij} + 1/f(s^{best}) \tag{1}$$

where $0 < \rho < 1$ is the pheromone decay parameter, and $f(s^{best})$ is the objective value of s^{best} in the current iteration. Global updating is intended to provide a greater amount of pheromone to good packing patterns.

Moreover, while ants construct a solution, a local pheromone updating rule is applied. The effect of local updating is to make the desirability of edges change dynamically. The local updating is made according to the following expression:

$$\tau_{ij} = (1 - \xi) \times \tau_{ij} + \xi \times \Delta\tau_{ij} \tag{2}$$

where $0 < \xi < 1$ is a parameter and $\Delta\tau_{ij}$ is set as τ_{min}. Dorigo and Gambardella [6] used this expression to run their experiments with good results.

Another way to promote exploration is by defining a lower limit (τ_{min}) for the pheromone values. The following formula sets the value of τ_{min} [10] as:

$$\tau_{min} = \frac{(1/(1-\rho))(1 - \sqrt[M]{pbest})}{(avg - 1)\sqrt[M]{pbest}} \tag{3}$$

where $pbest$ is the approximation of the real probability to construct the best solution, avg is the average number of pieces to choose from at every decision point when building a solution, defined as $M/2$. Also an evaporation phase occurs at each iteration by updating the pheromone trail by:

$$\tau_{ij} = \gamma \times \tau_{ij} \tag{4}$$

where $0 < \gamma < 1$ is a parameter.

3.3 State transition rule definition

The state transition rule gives the probability with which ant k will choose a piece j as the next piece for its current level l in the partial solution s,which is given by [10]:

$$j = \begin{cases} \max_{j \in J_k(s,l)} [\tau_l(j)] \times [\eta_j]^\beta & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases} \tag{5}$$

where $\tau_l(j)$ is the pheromone value for piece j in level l, η_j is the heuristic information guiding the ant, and β is a parameter which determines the relative importance of pheromone information versus heuristic information. q is a random number uniformly distributed in [0..1], q_0 is a constant parameter ($0 < q_0 < 1$) which determines the relative importance of exploitation versus exploration. S is a random variable selected according to the probability distribution given in Equation 6.

$$p_k(s,l,j) = \begin{cases} \frac{[\tau_l(j)]\times[\eta_j]^\beta}{\sum_{g\in J_k(s,l)}[\tau_l(g)]\times[\eta_g]^\beta} & \text{if } j \in J_k(s,l) \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

In Equations 5 and 6, $J_k(s,l)$ is the set of pieces that qualify for inclusion in the current level by ant k. The set includes those pieces that are still left after partial solution s is formed, and are light enough to fit in level l. The pheromone value $\tau_l(j)$ for a piece j in a level l is given by:

$$\tau_l(j) = \begin{cases} \frac{\sum_{i\in A_l}\tau_{ij}}{|A_l|} & \text{if } A_l \neq \emptyset \\ 1 & \text{otherwise} \end{cases} \quad (7)$$

where A_l is the set of current pieces allocated in level l. In other words, $\tau_l(j)$ is the sum of all the pheromone values of pieces already in level l, divided by the number of pieces in that level. This approach is similar to the one followed by Levine and Ducatelle [10].

3.4 The objective function

To assess the quality of the solutions, we first need to achieve the corresponding pieces layout. In order to generate a 3-stage level pattern, i.e., the pieces layout, we adopt a modified *next-fit decreasing height* heuristic (NFDH) —in the following referred as *modified next-fit*, or *MNF*— which was proven to be very efficient in [12, 15]. A more in-depth explanation of the MNF procedure can be found in [15].

The objective value of a solution s is defined as the strip length needed to build the corresponding packing pattern. An important consideration is that two packing patterns could have the same length —so their objective values will be equal— however, from the point of view of reusing the trim loss, one of them can be actually better because the trim loss in the last level (which still connects with the remainder of the strip) is greater than the one present in last level in the other layout. Therefore we use the following objective function:

$$f(s) = strip.length - \frac{l.waste}{strip.length * W} \quad (8)$$

where $strip.length$ is the length of the packing pattern corresponding to the permutation s and $l.waste$ is the area of reusable trim loss in the last level l of the packing pattern. Hence, function f is both simple and accurate.

3.5 The local search procedure

The local search procedure used in this work consists of the application of a modified version of first-fit decreasing heuristic (FFDH), called MFF, similar to the adjustment operator presented in a preliminary work [15]. In this way, the local search starts from a solution created by the ACS towards to the nearest local optimum for that solution, with the aim of improving the trim loss of all levels. After this improvement phase, the pheromone trail is updated, so it is in fact a form of Lamarckian search. A more in-depth explanation of the MFF procedure can be found in [15].

4 The external memory based ACS

In our proposed method, an external memory is maintained from which the ants acquire parts of solutions during the solution construction process [2]. This external memory stores L solution's sequences (partial permutations) of variable size taken from good solutions (packing patterns with small strip length) of previous iterations. In this case, the sequences are made up of k levels (set of pieces) from a solution, selected by considering their respective waste values. Since, we do not know in advance the number of pieces in the selected levels to be stored in the memory, each row of the memory must have the length of a complete solution (M). Hence, the dimensions of the external memory is $L \times M$, where L is the number of elements or rows in the memory and M is the number of pieces.

Each partial permutation has associated a lifetime value ($lifeTime$) and also the objective value ($cost$) of the solution from which it was extracted. The lifetime value initially is set to one, and is increased by 1 at each iteration. There is a maximum lifetime of partial permutation solution in the external memory, and it is set to 10 iterations. These values permit to assign a score to each partial permutation p ($1 \leq p \leq M$), which is calculated as $S(p) = cost(p) + lifeTime^2(p)$ [2]. This value is used by an ant to select a partial permutation when a new solution is to be created.

The ACS begins the evolutionary process with an empty external memory, which is fill immediately after some iterations performed by the ACS algorithm. The k best solutions are considered at each iteration and a number of levels are selected from that solutions and stored in the memory. At the early iterations the ACS fills the external memory. Notice that at this stage the ants build their solutions following only the classical probabilistic procedure of an ACS algorithm (as described in Section 3). Once the memory is full, the solution construction phase followed by an ants is modified. Under this implementation some ants constructs a solution by acquiring a part of existing solution from memory. In this case and for the problem under study, that part is a set of pieces that constitute a level in the packing pattern. The selection of the sequence from memory is made using tournament selection among T randomly selected partial permutation sequences, based on the score value assigned to each sequence. The winner is the best sequence with the lower score value. This sequence is copied in the first positions of the solution and in the meantime the local updating pheromone process takes place in exactly the same way it is done in normal ACS algorithm. Once this process finishes, the neighborhood has to be updated, deleting the pieces just incorporated in the solution. Until this point there are missing pieces in the partial solution. Therefore the remainder of the solution is constructed based on τ_{ij} as usual. In our implementation, there are a percentage of ants that continues building all their solutions with the tradicional scheme, this percentage is set to the 50%. The reason is to allow the ants build a complete new solution from the feedback of the pheromone trail and the heuristic information, which contributes to the exploration of new combination of pieces to devise new levels.

At the end of each iteration, the ACS updates the memory regarding the newest generated solutions. The k-best solutions generated by the ants at each iterations are considered to be donors. From each of these solutions, if l is the amount of levels in the solution, then the $l/2$ levels with the lowest waste are selected to be inserted in some entry of the external memory. In the case the memory is complete, a replacement strategy must to be applied. The selected element to be replaced is either the one with the worst objective value or the oldest one.

5 Implementation

Now we will comment on the actual implementation of a traditional ACS (as described in Section 3) and the ACSMem, a ACS algorithm which adds an external memory to help the ants during the solution construction process (as described in Section 4), to ensure that this work is replicable in the future. All these algorithms have been compared in terms of the quality of their results.

The number of ants is set to 50. Regarding their initial positioning, ants are placed randomly. The parameter values are the following: β= 2, q_0 = 0.9, ρ = 0.8, γ=0.96 and ξ=0.1. The initial pheromone value is set to τ_{min}. These parameters were used with success in [14]. The tournament size T is set to 7 and the best $k = 0.05 \times a$ ants solutions are used in the update memory phase. The number of memory rows L is fix to $a/2$.

The algorithms were implemented inside MALLBA [3], a C++ software library fostering rapid prototyping of hybrid and parallel algorithms. The platform was an Intel Pentium 4 at 2.4 GHz and 1GB RAM under SuSE Linux with 2.4.19-4 kernel version.

We have considered five randomly generated problem instances with M equal to 100, 150, 200, 250, and 300 pieces and a known global optimum equal to 200 (the minimum length of the strip). These instances belong to the subtype of level packing patterns but the optimum value does not correspond to a 3-stage guillotine pattern. They were obtained by an own implementation of a data set generator, following the ideas proposed in [16] with the length-to-width ratio of all M rectangles in the range $1/3 \leq l/w \leq 3$. These instances are publicly available at `http://mdk.ing.unlpam.edu.ar/~lisi/documentos/datos2spp.zip`.

6 Computational Analysis

In this section we summarize the results of applying the ACS and the ACSMem. Our aim is to offer meaningful results and check them from a statistical point of view. For each algorithm we have performed 30 independent runs per instance using the parameter values described in the previous section.

Table 1 shows the results of the ACS and ACSMem. The columns in this Table stand respectively for the best objective value obtained (*best*), the average objective values of the best found feasible solutions along with their standard deviations (*avg*$_{\pm\sigma}$), the average number of evaluations needed to reach the best value ($eval_b$) which represents the numerical effort, and the mean times (in seconds) spent in the full search (T_t). The minimum *best* values are printed in bold. From this table we can observe that both ACS algorithms present a similar performance in all instances. Hence, we can conclude nothing about the superiority of any version of the two models: they seem equally well

Table 1. Experimental Results.

Inst	ACS		ACSMem		*t*-test
	best	***avg***$_{\pm\sigma}$	***best***	***avg***$_{\pm\sigma}$	
100	215.78	218.29 $\pm$ 0.88	**215.00**	218.09 $\pm$ 0.91	-
150	**216.38**	217.82 $\pm$ 0.79	216.84	218.50 $\pm$ 0.97	+
200	**211.61**	214.37 $\pm$ 1.12	211.78	214.86 $\pm$ 1.18	-
250	**207.68**	209.20 $\pm$ 0.76	207.80	210.14 $\pm$ 0.70	+
300	213.66	214.74 $\pm$ 0.57	**212.45**	214.62 $\pm$ 0.94	-

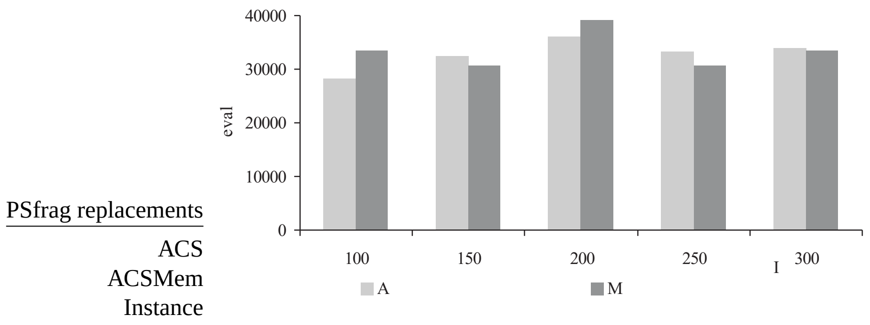

Fig. 1. Mean number of evaluations to reach the best value for each instance.

suited and efficient for the instances considered. This affirmation is statically corroborated more in the half of the instances, since there are no statistical significant differences between these options because the respective p-values for t-test are greater than the 0.05 significance level (see the "+ symbols meaning the significance of the t-test).

Now, we turn to the analysis of the number of evaluations, i.e., the numerical effort to solve the problem. Figure 1 shows the results of the ACS and ACSMem. Overall, it seems that the two algorithms need a similar effort to solve all the instances of the 2SPP (statistically corroborated by a t-test).

A remarkable aspect of the memory based ACS algorithm can be observed in Figure 2: the ACSMem produced a faster execution than the ACS. In fact, we can notice that there exists statistical confidence for this claim. The results can be explained because an ant begins the solution construction procedure from a partial permutation sequence copied from the external memory in ACSMem algorithms. The transfer process of the selected partial permutation from memory to the ants solution is faster than the probabilistic decisions needed to build that partial permutation.

Therefore, the lower execution times and the good results presented by ACSMem convert our proposal in a good approach to solve the 2SPP.

PSfrag replacements

ACS
ACSMem
Instance

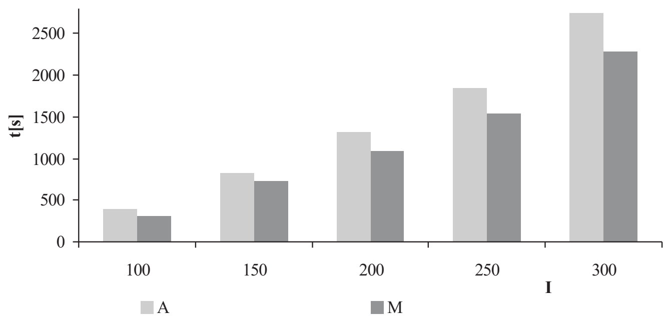

Fig. 2. Mean execution time for each instance.

7 Conclusions

In this paper we have presented a hybrid ACS using external memory to solve the 2SPP with additional constrains. The external memory store partial solutions in the form of group of pieces from elite solutions obtained in previous iterations. This partial solutions are selected regarding information from the layout instead of making the choice in a random way. In this particular case the group of pieces form a level with minimal waste. During solution construction, the ants copy part of the solution from the external memory, and after that they made the best possible choice, as indicated by the pheromone trail and heuristic information.

Computational results of the ACS with external memory are similar than those obtained with a traditional ACS. However, the incorporation of external memory helps to reduce the execution times of the ACS, since the spent time in the copy process of the partial solution already available from memory is smaller than the time inverted to construct that partial solution using probabilistic decisions as in the tradicional ACS.

There are several issues which seem to be worth for further investigation. One issue deals with the setting of parameters of the external memory implementation, such us the effects of different memory update strategies. Another issue can be the investigation of search space characteristics and their relation to the algorithm performance.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Science and Technology and the European FEDER under contract TIN2005-08818-C04-01 (the OPLINK project) and the proyect DIRICOM (P07-TIC-03044). We acknowledge the Universidad Nacional de La Pampa, Universidad Nacional de San Luis, and the ANPCYT in Argentina from which the first and third authors receive continuous support.

References

1. A. Acan. An external memory implementation in ant colony optimization. *Ant Colony Optimization and Swarm Intelligence (ANTS2004)*, pages 73–82, 2004.
2. A. Acan. An external partial permutation memory for ant colony optimization. *EvoCOP 2005, LNCS 3448*, pages 1–11, 2005.
3. E. Alba, J. Luna, L.M. Moreno, C. Pablos, J. Petit, A. Rojas, F. Xhafa, F. Almeida, M.J. Blesa, J. Cabeza, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, and C. León. *MALLBA: A Library of Skeletons for Combinatorial Optimisation*, volume 2400 of *LNCS*, pages 927–932. Springer, 2002.
4. M. Boschetti and V. Maniezzo. An ant system heuristic for the two-dimensional finite bin packing problem: preliminary results. *Chapter 7 of book Multidisciplinary Methods for Analysis Optimization and Control of Complex Systems*, pages 233–247, 2005.
5. M. Dorigo and L.M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43(2):73–81, 1997.
6. M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
7. J. Eggermont and T. Lenaerts. Non-stationary function optimization using evolutionary algorithms with a case-based memory. Technical Report 2001-11, Leiden University Advanced Computer Sceice (LIACS), 2001.
8. M. Guntsch and M. Middendorf. A population based approach for aco. *Applications of Evolutionary Computing - EvoWorkshops2002 LNCS 2279*, pages 72–81, 2002.
9. E. Hopper and B. Turton. A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artificial Intelligence Review*, 16:257–300, 2001.
10. J. Levine and F. Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, (55):705–716, 2004.
11. J. Montgomery and M. Randall. The accumulated experience ant colony for the travelling salesman problem. *International Journal of Computational Intelligence and Applications*, 3(2):189–198, 2003.
12. J. Puchinger and G. Raidl. An evolutionary algorithm for column generation in integer programming: An effective approach for 2d bin packing. In X. Yao et al, editor, *PPSN*, volume 3242 of *LNCS*, pages 642–651. Springer, 2004.
13. C. Ramsey and J. Grefenstette. Case-based initialization of gas. *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, 1993.
14. C. Salto, E. Alba, and J. M. Molina. *Optimization Techniques for Solving Complex Problems*, chapter Greedy Seeding and Problem-Specific Operators for GAs Solving Strip Packing Problems, pages 361–378. John Wiley & Sons, Inc., 2009.
15. C. Salto, J.M. Molina, and E. Alba. Evolutionary algorithms for the level strip packing problem. *Proceedings of NICSO*, pages 137–148, 2006.
16. P.Y. Wang and C.L. Valenzuela. Data set generation for rectangular placement problems. *EJOR*, 134:378–391, 2001.