Parallel ACO algorithms for 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 a parallel Ant Colony System (ACS) for the two-dimensional strip packing problem. In our computational study, we emphasize the influence of the incorporation of the received information in the target subcolony. Colonies send their best solutions instead of sending information from the matrix of pheromones, as happens in traditional parallel ACS. The solution arriving to a colony can provide further exploitation around promising solutions as this arrived solution can be used in both, the local update of the pheromone trail and the construction solution process of an ant. The aim of the paper is to report experimental results on the behavior of different types of parallel ACS algorithms, regarding solution qualities and parallel performance.

1 Introduction

All parallel ACO options studied so far in the literature have a common characteristic: the construction of a single solution by an ant is not split between multiple processors [8, 13]. The reason is that the solution construction process in ACO is typically a sequential process which is difficult to split in several independent parts. Consequently, the minimum grain size of parallel ACO is the construction of a single solution.

The majority of the parallel ACO algorithms assign more than one ant on each processor [2, 12]. When several ants are placed on a single processor and these ants work more closely than those ants in other processors, this group of ants are often called a *colony*. Those ACO algorithms that have several colonies of ants using their own pheromone matrix and where the pheromone matrices of different colonies are not necessarily equal are called *multicolony ACO algorithms*. This type of ACO algorithms are used in this work [8, 12]. Multicolony ACO algorithms are well suited for parallelization because a processor can host a colony of ants [1].

In this work we evaluate the application of three parallel ACO algorithms to solve a strip packing problem. We considered three parallel strategies to implement a parallel ant colony. In one of them, no communication is required between subcolonies. The other two strategies exchange information between the subcolonies. Traditional implementations exchange information of the pheromone matrix, but in this work the subcolonies send the best solution found-so-far to their neighbors [10]. The difference in these last two strategies is the way in which the information arriving to the target

subcolonies is used. All algorithms are studied from the numerical point of view but also from the parallel performance.

The article is organized as follows. Section 2 contains an explanation of the 2SPP. Section 3 describes the multicolony ACS used to solve the 2SPP. In Section 4, we explain the parameter settings of the algorithms used in the experimentation. Section 5 reports on the algorithm performances, and finally, in Section 6 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.

3 Parallel Ant Colony System to the 2SPP

Ant Colony System (ACS) [7, 6] 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.

Like many other metaheuristic approaches, the ACO metaheuristic admits direct parallelization schemes. Randall and Lewis [15] proposed an interesting classification of the parallelization strategies for ACO metaheuristic: parallel independent ant colonies, parallel interacting ant colonies, parallel ants, parallel evaluation of solution elements, and a combination of two of the mentioned strategies. In this work, we considered a version of the first and second strategies, which are described in the following paragraphs. The reason for this election is based on the good performance observed in the solution of other problems [2], and they performed similarly regarding the qualities of the results obtained.

In the case of parallel independent ant colony, there is a number of sequential ACSs which are put on different processors. This method, called $dACS_{ni}$, has the particularity that colonies do not send information. This alternative has a positive effect over

Algorithm 1 Algorithm dACS$_i$

```
init_pheromoneValues(τ); {Initialize the pheromone trails}
s_bs=build_solution(τ, η);
while not(stop condition) do
  for k ← 1 to μ do
    ant_k=build_solution(τ, η);{Ant ant_k builds their solution}
    localUpdate(τ,ant_k); {Local update of pheromone trials (ACS)}
    apply_localSearch(ant_k);
  end for
  if exchange_iteration then
    send\receive_solution(ant,dACS_j); {interaction with the neighborhood}
  end if
  feromone_evaporation(τ) {evaporation}
  for k ← 1 to μ do
    if f(ant_k) < f(s_bs) then
      s_bs=ant_k {actualize the best solution, daemon activity}
    end if
  end for
  globalUpdate(τ,ant,s_bs) {intensification of pheromone trails}
end while
return best solution found ant_bs
```

the behavior since each colony, which is running in a different processor, can specialize in different regions of the search space. The parallel interacting ant colonies strategies is similar to the previous one, except that an exchange of information between subcolonies occurs at a prefix iterations. The exchange of information is frequently associated to share the pheromone trail structure of the best performing colony among all the subcolonies. Also it is possible to send the best solutions found in each colony.

The distributed ACS algorithm that we use in this work is shown in Algorithm 1. The algorithm begins with the initialization of the pheromone trail associated with each transition. The principal loop of the algorithm consists of the following steps. A colony of μ ants incrementally build solutions (packing patterns) to the 2SPP applying a a stochastic local decision policy that makes use of pheromone trails and heuristic information. While the solution is being built, the ant deposits pheromone trails on the components or connections it used (local updating rule). This pheromone information will direct the search of future ants. Once the solution is created a local search procedure is activated. At preset iterations, the subcolony exchange information with the neighborhood (the communication structure used in this work corresponds to an unidirectional ring topology). After this communication step, a pheromone evaporation is triggered, which is the process by means of which the pheromone deposited by previous ants decreases over time. The best solution s_{bs} should be updated if an ant at the current iteration found a better packing pattern. Finally, a global pheromone update is carried out in order to deposit extra pheromone to good packing pattern. The algorithm returns the best solution found-so-far.

One of the aspects to consider in the moment of designing a dACS is the information exchanged between the colonies. One choice is to send solutions that have been found in a colony to its neighbor. Another choice is to send information from the matrix of pheromone. As the results of [10] indicate that the exchange of pheromone matrices is not desirable, in this work we have chosen to send the best solution found by the colony.

According to the unidirectional ring topology adopted in this work to communicate the different subcolonies, subcolony i influences the levels of pheromone trails of sub-

colony (i+1) mod n (where n is the number of subcolonies) by sending their best-so-far solution. In this work, the incoming solution is used in two different moments:

- *for local updating*. The solution arriving to a target subcolony is used in the local updating process together with the local set of solutions. Therefore, when only the best solution in a subcolony is allowed to update the pheromone values, the incoming solution influences the pheromone levels only when it is better than all the solutions found by ants in that iteration. This update made a very indirect use of the received information. This algorithm is referred as $dACS$ in the following.
- *for a more direct use of the information*. One way to complement the use of the received information from the neighboring colony is to use the information of the arrangement of the pieces of that solution in a more direct way. Therefore, we choose another alternative which consists in to extract the good levels of the incoming solution (those levels with a less waste) and to copy them to a pseudo-solution. This pseudo-solution is incomplete: some pieces are missed. The incoming solution is used in the local updating process as explained in previous paragraph and the process continues traditionally. In the next iteration of the algorithm, the ants begin the process of building their solution by copying the levels from the pseudo-solution and then they repeatedly apply a state transition rule to complete the packing pattern, using the pheromone trail and heuristic information. This combination allows a mix of exploitation (for the incoming solution) and exploration (for the experience of the target colony) through the respective pheromone matrix. The algorithm implementing this ideas is called $dACS_{mem}$.

The following paragraphs detail how the ACS can be applied to the 2SPP [16]. This description includes the most important elements of the ACS, namely the use of heuristic information, the pheromone trail definition, the state transition rule, and the local search procedure used in order to improve the solution quality.

We maintain solutions in the form of permutations of the set of pieces [5], which will be directly translated into the corresponding packing pattern by a layout algorithm. 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 [14, 18]. A more in-depth explanation of the MNF procedure can be found in [18].

The objective value of a solution s of ant$_s$ 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 the last level in the other layout. Therefore we use the following objective function:

$$f(s) = strip.length - \frac{l.waste}{strip.length * W} \quad (1)$$

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.

Heuristic definition. For the 2SPP, the problem-dependent heuristic information used is the height of piece j, i.e., the heuristic value for a piece j is $\eta_j = h_j$.

Pheromone definition. Trail τ_{ij} encodes the desirability of having a piece i and j in the same level [11]. 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).

Pheromone update. Once all ants have completed their packing patterns, a global pheromone updating rule is applied. In this case, only the best ant (which the respective solution is s_{bs}) is allowed to place pheromone after each iteration. This is done according to $\tau_{ij} = \rho \times \tau_{ij} + 1/f(s_{bs})$, where $0 < \rho < 1$ is the pheromone decay parameter, and $f(s_{bs})$ is the objective value of s_{bs}. Using only the best ant for updating makes the search much more aggressive. 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, which effect is to make the desirability of edges change dynamically. The local updating is made according to the following expression: $\tau_{ij} = (1 - \xi) \times \tau_{ij} + \xi \times \Delta\tau_{ij}$, where $0 < \xi < 1$ is a parameter and $\Delta\tau_{ij}$ is set as τ_{min}. Dorigo and Gambardella [7] 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} [11] as:

$$\tau_{min} = \frac{(1/(1-\rho))(1 - \sqrt[M]{pbest})}{(avg - 1)\sqrt[M]{pbest}} \tag{2}$$

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 $\tau_{ij} = \gamma \times \tau_{ij}$,

State transition rule definition. It 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 [11]:

$$j = \begin{cases} \max_{j \in J_k(s,l)} [\tau_l(j)] \times [\eta_j]^\beta & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases} \tag{3}$$

where $\tau_l(j)$ is the pheromone value for piece j in level l, η_j is the heuristic information guiding the ant, 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, and S is a random variable selected according to the probability distribution given in Equation 4.

$$p_k(s,l,j) = \begin{cases} \frac{[\tau_l(j)] \times [\eta_j]^\beta}{\sum_{g \in J_k(s,l)} [\tau_l(g)] \times [\eta_g]^\beta} & \text{if } j \in J_k(s,l) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In Equations 3 and 4, $J_k(s,l)$ 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:

$$\tau_l(j) = \begin{cases} \frac{\sum_{i \in A_l} \tau_{ij}}{|A_l|} & \text{if } A_l \neq \emptyset \\ 1 & \text{otherwise} \end{cases} \quad (5)$$

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 [11].

The local search procedure. It 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. The local search procedure used in this work consists of the application of a modified version of first-fit decreasing heuristic (FFDH), called MFF. A more in-depth explanation of the MFF procedure can be found in [18].

4 Implementation

Now we will comment on the actual implementation of the multicolony algorithms to solve the 2SPP: *i*) dACS with independent non-interacting colonies ($dACS_{ni}$) and two dACSs with colonies exchanging information: *ii*) $dACS$, which add the received solution to the solution set of the colony and *iii*) $dACS_{mem}$, which adds the received solution to the solution set as well as selects the best levels of the incoming solution with the objective that those levels will be used for ants in the next iterations.

The number of ants is set to 64, each subcolony has 64/n ants, where n represents the number of subcolonies. Each ant begins the building process of their solution with a piece randomly selected. 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 [17]. For the models involving communication between subcolonies, solutions were sent every 100 iterations following an asynchronous approach. Local search is applied to all solutions generated by the ants.

The algorithms were implemented inside MALLBA [3], a C++ software library fostering rapid prototyping of hybrid and parallel algorithms. The platform was a cluster of 16 PCs with Intel Pentium 4 at 2.4 GHz and 1GB RAM under SuSE Linux with 2.4.19-4 kernel version, and interconnected by a Fast-Ethernet at 100 Mbps.

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 [19] 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`.

5 Computational Analysis

In this section we summarize the results of applying the multicolony algorithms solve the 2SPP with restrictions, using different strategies to incorporate the information of

Table 1. Best fitness values for $ACSseq$ and the proposed dACS

Inst	$ACSseq$		$dACS_{ni}$		$dACS$		$dACS_{mem}$	
	best	***avg***$_{\pm\sigma}$	***best***	***avg***$_{\pm\sigma}$	***best***	***avg***$_{\pm\sigma}$	***best***	***avg***$_{\pm\sigma}$
100	215.78	218.29 $\pm$ 0.88	215.64	218.29 $\pm$ 0.86	**214.73**	217.93 $\pm$ 1.08	215.64	218.18 $\pm$ 0.98
150	216.38	217.82 $\pm$ 0.79	215.64	218.29 $\pm$ 0.86	215.69	217.82 $\pm$ 0.70	**214.79**	217.69 $\pm$ 0.88
200	211.61	214.37 $\pm$ 1.12	215.64	218.29 $\pm$ 0.86	210.77	213.29 $\pm$ 1.28	**210.68**	213.68 $\pm$ 1.18
250	207.68	209.20 $\pm$ 0.76	215.64	218.29 $\pm$ 0.86	207.70	209.28 $\pm$ 0.74	**207.54**	209.20 $\pm$ 0.62
300	213.66	214.74 $\pm$ 0.57	215.64	218.29 $\pm$ 0.86	**211.27**	213.98 $\pm$ 0.96	211.79	213.92 $\pm$ 0.68

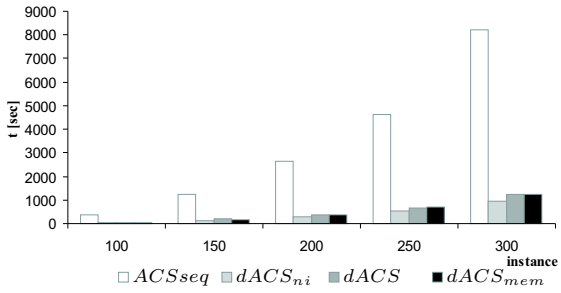

Fig. 1. Execution time for $seqACS$ and each dACS

the received solution. In a first place, a comparison of the multicolony algorithms is presented by establishing the same effort, a prefixed number of iterations. After that, the view point is changed in order to review the parallel performance. 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.

In order to obtain meaningful conclusions, we have performed an analysis of variance of the results. When the results followed a normal distribution, we used the t-test for the two-group case, and the ANOVA test to compare differences among three or more groups (multiple comparison test). We have considered a level of significance of $\alpha = 0.05$, in order to indicate a 95% confidence level in the results. When the results did not follow a normal distribution, we used the non-parametric Kruskal Wallis test (multiple comparison test), to distinguish meaningful differences among the means of the results for each algorithm.

Results with predefined effort. In this section, a sequential ACS, so-called $ACSseq$, is also included in the study, in order to show that the multicolony ACSs present not only lower runtimes but also better solutions to the problem. In order to make a fair comparison, all proposals stop after 65,536 evaluations (2^{32}). Table 1 shows the results of the different ACSs for each instance. The columns in this Table stand respectively for the best objective value obtained (*best*) and the average objective values of the best found feasible solutions along with their standard deviations ($avg_{\pm\sigma}$). The minimum *best* values are printed in bold.

Table 2. Quality of the target solutions to stop the algorithms

M	100	150	200	250	300
target	217.99	216.23	213.12	213.59	213.90

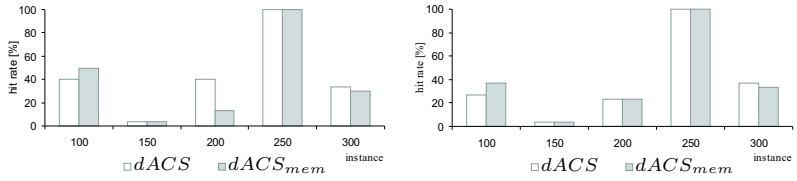

Fig. 2. Hit rates (in percentage) for each dACS algorithm: (a) 1 processor and (b) 8 processors

From this table we can observe that multicolony approaches are the algorithms that reach the best packing patterns for the whole set of instances; but there are no significant differences in the statistics analysis performed for instances with M=100, M=150 and M=250, i.e., the statistical tests indicate that all algorithms have presented similar mean values. The more important difference between the multicolony approaches and the sequential ACS is the run time, as to be expected (see Figura 1).

There are no differences between the multicolony approaches regarding the quality of the solution found, although a small advantage in favor of $dACS_{mem}$ is observed, since it solves more effectively three of the five instances. Regarding run times, $dACS$ and $dACS_{mem}$ present significant statistical differences only for instances with M=100 and M=250. From the examination of the mean run times values for those instances, it is observed that the differences are negligible, for example, $dACS$ took about 58.94 seg in the search meanwhile $dACS_{mem}$, 59.76 sec. in the instance with M=100, meaning a difference of 0.82 sec.; similar situation is present in the instance with M=250. This means that the additional processing incurred in saving the received solution and in extracting their good leves do not substantially affect the run time, which transforms this option in a viable alternative to obtain good solutions to 2SPP.

Results with predefined quality of solutions. Now we change the kind of analysis performed. We want to measure the time to find equivalent solutions with the dACSs proposed, in order to show their parallel characteristics. Thus we define our goal as reaching the fitness values which are shown in Table 2 . To carry out this experimentation, the eight subcolonies of each dACS are put on a same processor and then every subcolony is put in a dedicated processor.

Up to now, we have presented the average results over 30 independent runs. This time, we show the hit rate of the distributed ACSs, which is presented in Figure 2. This measure is the relation between the number of execution that reached the target fitness and the total number of performed tests. It is important to highlight that both $dACS$ and $dACS_{mem}$ reach the target value in all runs in instance with M=250, independently of the number of processors used. The eight sub-colonies running in sequence, i.e., using only one processor, obtain a similar hit rate that the parallel approaches, in instances with M=150 and M=300. Instance with M=150 has been difficult for any of the dACS

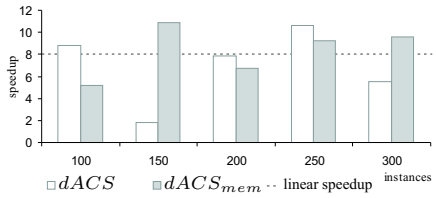

Fig. 3. Speedup values

algorithms, since the hit rate is equal to 3.3%, meaning that in only one run from the 30 the solution obtained was better than the target.

From the point of view of the parallel performance of the algorithms, Figure 3 shows the speedup values, which are obtained following the orthodox definition of speedup [4]. We can see high speedup values in the majority of the dACS approaches, except for $dACS$ and the instance with M=150, where the speedup value is lower than two. In particular, superlinear speedups are observed in both algorithms in instance with M=250. These results suggest that we are using good parallel implementations of the algorithms.

6 Conclusions

In this paper we have presented different parallel ACSs to solve the 2SPP with additional constrains. The parallelization strategy consisted in a multicolony model, where sporadic exchange of solutions between subcolonies occurs. Therefore, the exchange of solutions between subcolonies can be considered as a class of interaction among parallel ant colonies. The characteristics of the distributed models have proven to be good techniques to obtain good packing patterns, which represents a great step forward in this field.

Computational results of the three considered multicolony strategies are similar than those obtained with a sequential ACS, but $dACS_{mem}$ and $dACS$ obtained the best packing patterns. The most important difference between the sequential ACS and the multicolony ACSs was observed in the run times: the last ones reduced the time involved in the search. The results suggest that the exchange of information in the proposed dACSs do not help the search, similar conclusion are reported in [2].

There are several issues which seem to be worth for further investigation. One issue deals with the effects of different uses of the information of solution arriving to promote a more direct use of the information in the algorithm. 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 second authors receive continuous support.

References

1. E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, 2005.
2. E. Alba, G. Leguizamon, and G. Ordoez. Two models of parallel ACO for the minimum tardy task problem. *Int. Journal High Performance Systems Architecture*, 1:50–59, 2007.
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. E. Alba and J. M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Comput. Systems*, 17:451465, 2001.
5. 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.
6. M. Dorigo and L.M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43(2):73–81, 1997.
7. 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.
8. L.M. Gambardella, E. Taillard, and M. Dorigo. *New Ideas in Optimization*, chapter MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows, pages 63–76. McGraw-Hill, 1999.
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. F. Kruger, D. Merkle, and M. Middendorf. Studies on a parallel ant system for the BSP model. 1998.
11. 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.
12. R. Michels and M. Middendorf. *New Ideas in Optimization*, chapter An ant system for the shortest common supersequence problem, pages 51–61. McGraw-Hill, 1999.
13. M. Middendorf, F. Reischle, and H. Schmeck. Multicolony ant system algorithms. *Journal of Heuristics (Special issue on Parallel Metaheuristics*, 8(3):305–320, 2002.
14. 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.
15. M. Randall and A. Lewis. A parallel implementation of ant colony optimization. *Journal of Parallel and Distributed Computing*, 62:1421–1432, 2002.
16. C. Salto, E. Alba, and J. M. Molina. Hybrid ant colony system to solve a 2-dimensional strip packing problem. *International Conference on Hybrid Intelligent Systems*, pages 708–713, 2008.
17. 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.
18. C. Salto, J.M. Molina, and E. Alba. Evolutionary algorithms for the level strip packing problem. *Proceedings of NICSO*, pages 137–148, 2006.
19. P.Y. Wang and C.L. Valenzuela. Data set generation for rectangular placement problems. *EJOR*, 134:378–391, 2001.