A hybrid improvement heuristic for the bin-packing problem and its application to the multiprocessor scheduling problem

Adriana C. F. Alvim[1*], **Celso C. Ribeiro**[1]

[1]Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente 225 – 22453-900 Rio de Janeiro, RJ

alvim@inf.puc-rio.br, celso@inf.puc-rio.br

Abstract. *This work presents a hybrid improvement procedure for the bin packing problem and its application to the multiprocessor scheduling problem. This heuristic has several features: a new lower bound; reductions; initial solutions by reference to the dual problem; heuristics for load redistribution, and an improvement process utilizing tabu search. It improved the best known solutions for many of the benchmark instances and found the largest number of optimal solutions with respect to the other available approximate algorithms.*

1. Introduction

Given a set $N = \{1, \dots, n\}$ of items with weights $w_i \, (i = 1, \dots, n)$, the bin packing (BP) problem consists of finding the minimum number m of bins of capacity C necessary to pack the items without violating the capacity constraints. BP is known to be closely related to the multiprocessor scheduling problem ($P||C_{\max}$), which is the problem of scheduling n independent tasks with associated processing times w_i on m parallel identical processors with the objective of minimizing the maximum completion time of a task (makespan). Both problems are NP-hard [Garey and Johnson, 1979]. They share the same decision problem, which is to determine whether all items/tasks can be assigned to m bins/processors with bin capacity/makespan equal to C. This kind of dual relationship [Hochbaum and Shmoys, 1987, Scholl et al., 1997] is explored in this work.

The branch-and-bound procedure MTP [Martello and Toth, 1990] is a basic reference for BP. Another branch-and-bound procedure was proposed by Scholl et al. (1997). Valério de Carvalho (1999) presented an exact algorithm based on colunm generation. Schwerin and Wäscher (1999) proposed the procedure MTPCS which combines MTP with the bound L_{CS} derived from the cutting stock problem. Recently, Fleszar and Hindi (2002) proposed a few new heuristics to BP. Dell'Amico and Martello (1995) developed a branch-and-bound algorithm to exactly solve $P||C_{\max}$.

In this work, we present a hybrid improvement procedure for the bin packing problem (HI_BP) and its application to the multiprocessor scheduling problem (HI_PCmax). A new lower bound for BP is presented in Section 2. The full hybrid improvement heuristic and its three phases is described in Section 3. Computational results, comparisons with other heuristics for different classes of test problems and concluding remarks are presented in Section 4.

[*]Supported by Pontifícia Universidade Católica do Rio de Janeiro, CNPq and CAPES.

2. Lower Bound L_ϑ

A trivial lower bound to BP is given by $L_1 = \lceil \sum_{i=1}^{n} w_i/C \rceil$. We propose a new *destructive bound* [Scholl et al., 1997] for BP, based on the work of Dell'Amico and Martello (1995) for $P||C_{\max}$. We assume that the items are sorted in non-increasing order of their weights. Given a lower bound m to the number of bins in a feasible solution, we attempt to establish that no feasible solution using m bins exist, in which case this bound can be increased by one. Let $\theta = \max_{q=1,\ldots,n}\{q : \sum_{i=n-q+1}^{n} w_i \le C\}$ be an upper bound to the number of items in any bin of a feasible solution to BP. The proofs of the following propositions and theorems are given in Alvim (2003).

Proposition 1 *If $\theta < \lceil n/m \rceil$ for some integer m, then any feasible solution to BP uses at least $m+1$ bins.*

Proposition 2 *Given an integer $\sigma \le \lfloor n/m \rfloor$, if $C(\sigma) = \lceil \sum_{i=\sigma}^{n} \frac{w_i}{m-1} \rceil > C$, then any feasible solution with m bins has at least σ items in each bin.*

Proposition 3 *Given an integer $\sigma \le \lfloor n/m \rfloor$, if $\sum_{i=1}^{\sigma} w_i \le C$ then there exists an optimal solution using m bins having at least σ items in each bin.*

The next theorem follows directly from Propositions 2 and 3:

Theorem 1

$$\vartheta = \max\{\max_{\sigma=1,\ldots,\lfloor n/m \rfloor}\{\sigma : C(\sigma) > C\}, \max_{\sigma=1,\ldots,\lfloor n/m \rfloor}\{\sigma : \sum_{i=1}^{\sigma} w_i \le C\}\}$$

is a lower bound to the number of items in any bin of a feasible solution using m bins.

The next two propositions show how the lower bound ϑ and the upper bound θ to the number of items in any bin of any solution with m bins can be used to improve the lower bound to the number of bins.

Proposition 4 *Let $\underline{m}_\vartheta = \max\{m - (n - \vartheta \cdot m), 0\}$ be a lower bound to the number of bins with exactly ϑ items. If $\lceil \sum_{i=\underline{m}_\vartheta \cdot \vartheta + 1}^{n} \frac{w_i}{(m - \underline{m}_\vartheta)} \rceil > C$, then $m+1$ is a valid lower bound to the number of bins in the optimal solution.*

If $\theta = \vartheta + 1$, each bin has either ϑ or θ items. If this is the case, the number of m_ϑ of bins with exactly ϑ items and the number m_θ of bins with exactly θ items may be computed as the solution of the system $(m = m_\theta + m_\vartheta, n = \theta \cdot m_\theta + \vartheta \cdot m_\vartheta)$ yielding $m_\vartheta = (\vartheta + 1) \cdot m - n$ and $m_\theta = n - \vartheta \cdot m$. Then, a lower bound to the capacity needed to accommodate the n items in m bins is given by

$$C_\vartheta(m) = \max\left\{\left\lceil \sum_{i=n-\vartheta \cdot m_\vartheta + 1}^{n} w_i/m_\vartheta \right\rceil, \left\lceil \sum_{i=n-\theta \cdot m_\theta + 1}^{n} w_i/m_\theta \right\rceil\right\}.$$

Proposition 5 *If $\theta = \vartheta + 1$ and $C_\vartheta(m) > C$, then $m+1$ is a valid lower bound to the number of bins in the optimal solution.*

The next theorem follows directly from Propositions 1, 4, and 5:

Theorem 2 *Let m be a lower bound to the number of bins. Then,*

$$L_\vartheta(m) = \begin{cases} m+1, & \textit{if any of the conditions (a), (b), or (c) below holds:} \\ & \textit{(a) } \theta < \lceil n/m \rceil \\ & \textit{(b) } \lceil \sum_{i=\underline{m}_\vartheta \vartheta+1}^{n} \frac{w_i}{(m-\underline{m}_\vartheta)} > C \rceil \\ & \textit{(c) } \theta = \vartheta + 1 \textit{ and } C_\vartheta(m) > C; \\ m, & \textit{otherwise} \end{cases}$$

is an improved valid lower bound to the number of bins in a feasible solution.

3. The two hybrid improvement heuristics: HI_BP and HI_PCmax

We summarized below the basic steps of HI_BP and HI_PCmax. Each procedure has one specific preprocessing step and share the same core procedure formed by the construction, redistribution and improvement phases, which use, in addition to the problem data, the parameters: target value for the number of bins ($target_m$) and target value for the makespan ($target_C$). A solution is feasible to BP if the makespan is no greater than $target_C$ (bin capacity). A solution is feasible to $P||C_{\max}$ if it uses no more than $target_m$ bins.

- Preprocessing for HI_BP, given n items with weights w_i and bin capacity C: Use reduction procedure MTRP [Martello and Toth, 1990]. Compute an upper bound ub_m to the number of bins m. Compute lower bounds to m: L_3 by Martello and Toth (1990), $L_*^{(20)}$ by Fekete and Schepers (2001) and L_ϑ (Section 2). Set $target_C \leftarrow C$ and $target_m \leftarrow \max\{L_3, L_*^{(20)}, L_\vartheta\}$. If $target_m = ub_m$, then stop.
- Preprocessing for HI_PCmax, given n tasks with processing times w_i and m processors: Compute upper bound ub_C to the makespan C. Compute lower bounds to C: L_3, L_{HS} and L_ϑ as described by Dell'Amico and Martello (1995). Set $target_m \leftarrow m$ and $target_C \leftarrow \max\{L_3, L_{HS}, L_\vartheta\}$. If $target_C = ub_C$, then stop.
- Core procedure, given $n, w_i, target_C$ and $target_m$:
  - Construction: build a feasible solution to $P||C_{\max}$.
  - Redistribution: if the current solution is not feasible to BP, then apply load balancing/unbalancing strategies to improve bin usability.
  - Improvement: if the current solution is not feasible to BP, then use a tabu search heuristic to attempt to knock down capacity violations.
- Stopping criterion: if the current solution is feasible to BP, then stop; otherwise HI_BP sets $target_m \leftarrow target_m + 1$ and HI_PCmax sets $target_C \leftarrow target_C + 1$; go back to the core procedure.

3.1. Construction phase and upper bounds

Let S be a (not necessarily feasible) solution to BP with m bins. Associated with solution S there is a family of subsets $B_1, \ldots, B_m$ where B_j is used to denote both the j-th bin itself and the set of items it contains, for every $j = 1, \ldots, m$. Let $w_S(j) = \sum_{i \in B_j} w_i$ be the total weight of the items placed in bin B_j in solution $S, j = 1, \ldots, m$. Then, each bin B_j is in exactly one of the following situations: *violated* ($w_S(j) > C$), *complete* ($w_S(j) = C$), *incomplete* ($C > w_S(j) > 0$) or *empty* ($w_S(j) = 0$). A bin is *saturated* if it is violated or complete. Bins which are not violated are said to be *feasible*.

The following construction heuristics were used for building feasible solutions and upper bounds to $P||C_{\max}$. All of them start with m open bins and investigate the items in non-increasing order of their weights.

- Dual Best-Fit Decreasing (DBFD): select the bin with smallest sufficient residual capacity; if none is available then select the lightest bin.
- Dual Best 3-Fit Decreasing (DB3FD): if there is an empty bin, then select it to place the current item. Otherwise, perform an attempt to fill exactly each bin, by identifying a pair of yet unselected items whose sum of their weights is equal to the residual capacity of the bin. For the remaining unselected items, insert the current item into the heaviest non-saturated bin in which it fits; if none is available then the lightest bin is selected.
- Dual Worst-Sum-Fit Decreasing (DSSFD): if there is an empty bin, then insert the current item into this bin and perform an attempt to fill it by solving a subset sum problem (using heuristic MTSS of Martello and Toth (1990)). Otherwise, the current item is inserted into the lightest bin.
- LPT: this is the Longest Processing Time heuristic by Graham (1969).
- AHS: this is the the $(1/5 + 2^{-k})$-approximation algorithm for the minimum makespan problem proposed by Hochbaum and Shmoys (1987).

Two of the fastest heuristics for the approximate solution of BP are the well-known First-Fit Decreasing (FFD) and Best-Fit Decreasing (BFD) greedy algorithms, see e.g. Martello and Toth (1990) for a review. HI_BP uses BFD in the preprocessing step and heuristics DBFD, DB3FD, DWSFD and LPT in the construction phase. HI_PCmax uses heuristics LPT, AHS and DWSFD in the preprocessing and construction phases.

3.2. Redistribution phase

Whenever a feasible solution to $P||C_{\max}$ is not feasible to BP, load balancing and load unbalancing sub-strategies are applied to improve bin usability by load redistribution.

Load balancing: given any pair of bins of the current infeasible solution S, a new solution S' can be obtained by redistributing the items in these bins, so as to minimize the absolute value of the difference of their weights. Since the latter amounts to a number partitioning problem, an approximate algorithm provides a significant efficiency advantage for computing a suboptimal redistribution of the items in these two bins. The differencing method of Karmarkar and Karp (1982)is applied to all pairs of bins of the current solution, in which one of them is *violated* and the other is *non-saturated* to more nearly equalize the weights of items assigned to these restricted pairs of bins and minimize the maximum deviation. The search stops when no further improvement is possible. This procedure can make an infeasible solution feasible.

Load unbalancing: given n items with weights w_i, and an integer C, the maximum subset sum problem consists of finding a feasible subset of items whose sum of their weights is as close as possible to C. For every pair of *incomplete* bins in the current solution S, the load unbalancing sub-strategy attempts to redistribute their items without making them infeasible and creating more available space in the bin which ends up as the lighter among the two bins. We create a temporary set of available items, formed by all the items in this pair of bins, and apply the polynomial-time approximation scheme MTSS(3) of Martello and Toth (1990). If the sum of the weights of the subset found by the above algorithm is greater than the weight of the heaviest bin originally in the pair, then the composition of the two original bins is changed and a new solution S' is obtained. One bin receives all items in the solution of the subset sum problem, while the other receives the remaining items. The search stops when no further improvement is possible, after all pairs of incomplete bins have been evaluated. Although this procedure cannot make an infeasible solution feasible, it makes the current solution more susceptible to improvement in the next phase by creating more available space for large items.

3.3. Improvement phase

We apply a tabu search strategy to reduce capacity violations in the current solution. For any solution S, we denote by $E_S = \sum_{j=1}^{m} \max\{0, w_S(j) - C\}$ the sum of all bin capacity violations. $E_S = 0$ if S is feasible to BP. Starting from an infeasible solution S, we investigate neighborhoods defined by swap moves which exchange pairs of items, one of them always from a $violated$ bin. For any item $i = 1, \ldots, n$, we denote by $S(i)$ the index of the bin were this item is currently placed in solution S. Each move $i \leftrightarrow k$ is defined by an ordered pair (i, k) of items from different bins. The first element in the pair is always an item in the target *violated* bin, whose excess deviation we want to reduce. The solution S' resulting from applying this move to solution S is characterized by $S'(i) = S(k)$, $S'(k) = S(i)$, $S'(\ell) = S(\ell)\ \forall \ell \neq i, k$. We only consider moves that decrease the excess deviation of the target violated bin, i.e., swap moves for which $w_i > w_k$. The value $\Delta(i, k) = \max\{w_{S'}(S'(i)) - C, 0\} + \max\{w_{S'}(S'(k)) - C, 0\}$ gives the excess violation associated exclusively with the bins where these items are placed after their exchange.

We consider six types of moves, one for each possible combination of the bins situation after a move $i \leftrightarrow k$: (1) *complete* and *complete*, (2) *complete* and *incomplete*, (3) *incomplete* and *incomplete*, (4) *complete* and *violated*, (5) *incomplete* and *violated* and (6) *violated* and *violated*. For each move type we consider three criteria: (i) the associated move value $\Delta(i, k)$ (≥ 0), (ii) the number of violated bins ($0, 1$ or 2) and (iii) the number of complete bins ($0, 1$ or 2). We now discuss what qualifies a candidate move as being better than another. In principle, moves leading to pairs with less violated bins are preferable. Although less important, moves leading to pairs with more complete bins are also preferable. The proposed strategy categorizes each possible move by a "priority". For a given target violated bin whose neighborhood is being investigated, one chooses the move with the higher priority, breaking ties in favor of the bin with the smaller excess deviation $\Delta(i, k)$. Inasmuch as it may not be possible to establish a total order over all possible moves (since one move type may be better than another with respect to one criterion, but not with respect to the other), two different potential priority values are assigned to some move types and one of them is randomly selected with probability 1/2 at each iteration. For example: a move of type (5), leading to one violated bin with $\Delta(i, k) = 3$, is better or worst than a move of type (6), leading to two violated bins with $\Delta(i, k) = 2$? It is not clear. The proposed rules allow different choices at different iterations, avoiding inflexible preferences that could exclude some search paths. These rules are particularly useful in the context of a solution method which accepts moves leading to infeasible solutions that may eventually be made feasible at a later step. Another tabu search feature used in this work is logical restructuring based on anticipatory analysis [Glover and Laguna, 1997].

Whenever a move $i \leftrightarrow k$ is performed, we forbid for a duration of `TabuTenure` iterations all moves that would reinsert either item k into bin $S(k)$ or item i into bin $S(i)$, where `TabuTenure` is randomly chosen from a discrete uniform distribution in the interval $[0.8 \cdot \sqrt{n}, 1.2 \cdot \sqrt{n}]$. The improvement phase stops if a feasible solution

Table 1: HI_BP vs. Perturbation MBS' + VNS [Fleszar and Hindi, 2002]

		HI_BP			Perturbation MBS' + VNS		
Group	#	opt	max abs dev	time (s)	opt	max abs dev	time (s)
Group-I	160	160	0	0.52	159	1	0.03
Group-II	1210	1210	0	0.15	1170	2	0.16
Total	1370	1370	0	0.19	1329	2	0.14

is found or if a total of `MaxIterations` tabu search iterations have been performed without improvement in the total excess violation.

4. Computational experiments and concluding remarks

We report computational experiments for BP and for $P||C_{\max}$. Heuristic HI_BP sets `MaxIterations`=4000 and HI_PCmax sets `MaxIterations` = 1000. They were coded in `C` and compiled with version 2.95.2 of `gcc`, using the optimization flag -O3. Experiments were performed on a 1.7 GHz Pentium IV, 256 Mbytes of RAM memory.

We consider four groups of test problems for BP. Group-I, distributed by the OR Library (http://mscmga.ms.ic.ac.uk/jeb/orlib/binpackinfo.html), has 160 instances. Group-II is composed by 1210 instances available from Scholl and Klein (2003). Group-III is composed by 217 instances distributed by SICUP (2003) and Group-IV is composed by 100 instances of each of 145 ffd-hard and extremely-ffd-hard classes proposed by Schwerin and Wäscher (1997). We compare our results with those obtained by Perturbation MBS'+VNS [Fleszar and Hindi, 2002] on a 400 MHz Pentium, for 1370 instances (Group-I and Group-II). Table 1 gives, for each group, the number of instances and, for each heuristic, the number of instances for which the optimal solution was found, the maximum absolute deviation, and the average computation times in seconds. HI_BP found optimal solutions for all instances, 41 additional instances than Perturbation MBS'+VNS. HI_BP also solved four open instances from Group-II, as reported by Scholl and Klein (2003). We also compared our results with those reported by SICUP (2003) for the instances of Group-III. HI_BP improved the best results for 11 instances and found the same results for the others. HI_BP benefits of using strong lower bounds, as the search stops as soon as it identifies a proven optimal solution. We remark that the new lower bound L_θ identified 18 optimal solutions of Group-II and 37 optimal solutions of Group-III. An extension of this work [Alvim et al., 2004] compared our results with those obtained by MTPCS [Schwerin and Wäscher, 1999] for Group-IV. HI_BP performed consistently better than MTPCS, solving to optimality 97.9% of the 14500 instances against 84.2% by MTPCS.

We considered two groups of test problems for $P||C_{\max}$. Each group is formed by ten test problems for each of 39 classes. These classes are characterized by different combinations of $m \in \{5, 10, 25\}$ and $n \in \{10, 50, 100, 500, 1000\}$ with processing times randomly generated in the intervals [1,100], [1,1000], and [1,10000]. The two groups differ by the distribution of the processing times: uniform [França et al., 1994] and non-uniform [Necciari, 2001]. We compared the new heuristic HI_PCmax [Alvim and Ribeiro, 2004] with the branch-and-bound code (B&B) by Dell'Amico and Martello (1995) with the number of backtracks set at 4000. Table 2 summarizes the main results obtained by algorithms HI_PCmax and B&B on the same

Table 2: Comparative results: HI_PCmax vs. B&B.

		HI_PCmax				B&B			
		opt	max abs	rel avg	time	opt	max abs	rel avg	time
Group	$t_i \in$		error	error	(s)		error	error	(s)
uniform	$[1, 100]$	130	0	0.0000	0	130	0	0.0000	0
	$[1, 1000]$	126	1	1.36e-05	0.02	126	3	2.22e-05	0.03
	$[1, 10000]$	110	12	3.46e-05	0.15	107	173	4.33e-04	0.17
non-uniform	$[1, 100]$	120	7	6.79e-04	0.14	87	20	2.12e-03	2.12
	$[1, 1000]$	128	25	1.03e-04	0.20	79	152	1.77e-03	1.52
	$[1, 10000]$	121	253	1.04e-04	0.72	77	880	1.80e-03	3.99

computational environment. For each group of test problems and for each algorithm, it indicates the number of optimal solutions found over the 130 instances, the maximum absolute errors (w.r.t. the best lower bound), the average relative errors, and the average computation times in seconds. The superiority of HI_PCmax is clear for the non-uniform instances. It not only found better solutions, but also in smaller computation times.

Another experiment was done to investigate the effectiveness of the preprocessing, construction, redistribution and improvement phases of HI_BP and HI_PCmax. The main results showed that: (i) the redistribution phase alone is a good heuristic for BP (472 optimal solutions out of 581); (ii) the construction heuristic DWSFD revealed itself as a very effective approximate algorithm for $P||C_{\max}$ (556 optimal solutions out of 780); (iii) the improvement phase is essential to solve difficult instances.

The effectiveness of our hybrid improvement procedure is mainly due to the combination of several features: reduction, use of several known and new lower bounds, initial solutions by reference to the dual problem; heuristics for load redistribution based on differencing and unbalancing; and an improvement process utilizing tabu search. The move selection strategy used by the tabu search is a major contribution of this work and very likely can be applied to other problems in similar situations. HI_BP and HI_PCmax compare favorably with other approaches. We note that the best results previously reported in the literature for BP were not all of them obtained by a single heuristic. Although other heuristics were able to find similar results for some classes of test problems, HI_BP found the largest number of optimal solutions with respect to the other available approximate algorithms. The improvement heuristic HI_PCmax outperformed the other approximate algorithms in the literature, in terms of solution quality and computation times.

We conclude that the main contributions of this work are as follows: a new lower bound for BP; a good load redistribution heuristic for BP; the move selection strategy used by the tabu search; a good construction heuristic for $P||C_{\max}$ and two efficient and robust improvement heuristics for the bin packing and the multiprocessor scheduling problems.

References

Alvim, A. C. F. (2003). *Uma Heurística Híbrida de Melhoria para o Problema de Bin Packing e sua Aplicação ao Problema de Escalonamento de Tarefas*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro.

Alvim, A. C. F. and Ribeiro, C. C. (2004). A hybrid bin-packing heuristic to multiprocessor scheduling. In Ribeiro, C. and Martins, S., editors, *Lecture Notes in Computer Science*, volume 3059, pages 1–13. Springer-Verlag.

Alvim, A. C. F., Ribeiro, C. C., Glover, F., and Aloise, D. J. (2004). A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, 10:205–229.

Dell'Amico, M. and Martello, S. (1995). Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing*, 7:191–200.

Fekete, S. P. and Schepers, J. (2001). New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91:11–31.

Fleszar, K. and Hindi, K. S. (2002). New heuristics for one-dimensional bin packing. *Computers and Operations Research*, 29:821–839.

França, P. M., Gendreau, M., Laporte, G., and Müller, F. M. (1994). A composite heuristic for the identical parallel machine scheduling problem with minimum makespan objective. *Computers and Operations Research*, 21:205–210.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Boston.

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17:416–429.

Hochbaum, D. S. and Shmoys, D. B. (1987). Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of ACM*, 34:144–162.

Karmarkar, N. and Karp, R. M. (1982). The differencing method of set partitioning. Relatório Técnico UCB/CSD 82/113, Computer Science Division, University of California, Berkeley.

Martello, S. and Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. Wiley, London.

Necciari, E. (2001). http://www.di.unipi.it/di/groups/optimize/Data/MS.html.

Scholl, A. and Klein, R. (2003). http://www.wiwi.uni-jena.de/Entscheidung/binpp/.

Scholl, A., Klein, R., and Jürgens, C. (1997). BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers and Operations Research*, 24:627–645.

Schwerin, P. and Wäscher, G. (1997). The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research*, 4:337–389.

Schwerin, P. and Wäscher, G. (1999). A new lower bound for the bin-packing problem and its integration to MTP. *Pesquisa Operacional*, 19:111–129.

SICUP (2003). http://www.apdio.pt/sicup/Sicuphomepage/research.htm.

Valério de Carvalho, J. M. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659.