A HYBRID IMPROVEMENT HEURISTIC FOR THE ONE-DIMENSIONAL BIN PACKING PROBLEM

ADRIANA C.F. ALVIM, CELSO C. RIBEIRO, FRED GLOVER, AND DARIO J. ALOISE

Abstract. We propose in this work a hybrid improvement procedure for the bin packing problem. This heuristic has several features: the use of lower bounding strategies; the generation of initial solutions by reference to the dual min-max problem; the use of load redistribution based on dominance, differencing, and unbalancing; and the inclusion of an improvement process utilizing tabu search. Encouraging results have been obtained for a very wide range of benchmark instances, illustrating the robustness of the algorithm. The hybrid improvement procedure compares favourably with all other heuristics in the literature. 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 of bins of capacity C necessary to pack the items without violating the capacity constraints. Alternatively, the problem may also be seen as that of partitioning the set of items into a minimum number of subsets, such that the sum of the weights of the items in each subset is less than or equal to C. This problem is classified as $1/V/I/M$ according with the typology of Dyckhoff [6]. This formulation also entails a min-max problem sharing a kind of duality relationship [16, 24] with BP, in which we seek to minimize the capacity C of a fixed number of identical bins, so that all items can be accommodated in these bins without violating the capacity constraints. The dual bin packing problem (DBP) is known in the literature as the Multiprocessor Scheduling Problem. This relationship is explored by our heuristic for the bin packing problem.

The bin packing problem is NP-hard [10, 19]. The branch-and-bound procedure MTP of Martello and Toth [20] is the basic reference used in most comparative studies. Scholl et al. [24] proposed an exact method (BISON) which makes use of several bounds, reduction procedures, heuristics, and a branch-and-bound procedure using a new branching scheme. Later, Schwerin and Wäscher [27] showed that MTP provides significantly better results using the bound L_{CS} derived from the cutting stock problem. Valério de Carvalho [30] presented an exact algorithm based on colunm generation and branch-and-bound. Vanderbeck [31] proposed yet another column generation based exact algorithm for the cutting stock problem and showed its effectiveness for some classes of bin packing instances. 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. [20]

for a review. Hübscher and Glover [16] proposed a tabu search with influential diversification algorithm for DBP. Falkenauer [7] described a hybrid grouping genetic algorithm (HGGA) for BP. More recently, Fleszar and Hindi [9] proposed a few new heuristics to BP, the most effective of them being based on the VNS metaheuristic [15] and using new lower bounds proposed by Fekete and Schepers [8].

In this work, we describe a hybrid improvement procedure for the bin packing problem, originally outlined in [1, 2] and based on the progressive increase of the number of bins used by a feasible solution to DBP. This type of strategy is also known as a "lower bound method" in the assembly line balancing literature, see e.g. Hackman et al. [14] and Scholl and Voss [25]. The basic structure of this procedure is the following:

- *Reductions:* use reduction techniques to eliminate some items and to fix the items in some bins.
- *Bounds:* compute lower and upper bounds LB and UB for BP. If $LB = UB$, then stop.
- *Construction:* apply a greedy algorithm to build a solution for DBP using exactly LB bins.
- *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 set $LB \leftarrow LB + 1$ and go back to the construction phase.

The reduction procedure of Martello and Toth [21] is summarized in Section 2. Lower bounds and greedy construction procedures are briefly reviewed in Section 3. In Section 4, we present different construction algorithms used to build initial solutions for the hybrid improvement procedure. Section 5 describes an item dominance rule and the load balancing/unbalancing strategies used to improve bin usability. A tabu search procedure to reduce infeasibilities is proposed in Section 6, and the full hybrid improvement heuristic is described in Section 7. Computational results and comparisons with other heuristics for different classes of test problems are presented in Section 8. Some concluding remarks and extensions are discussed in the last section.

2. Reductions

We define a feasible set of items as any subset $F \subseteq N$ such that $\sum_{i \in F} w_i \leq C$. Given two feasible sets F_1 and F_2, we say that F_1 dominates F_2 if and only if the number of bins in some optimal solution obtained by setting $B_1 = F_1$ is not greater than that obtained by setting $B_1 = F_2$. This will be the case if there exists a partition $P_1, \ldots, P_\ell$ of F_2 and a subset $\{i_1, \ldots, i_\ell\} \subset F_1$ such that $w_{i_h} \geq \sum_{k \in P_h} w_k$ for $h = 1, \ldots, \ell$.

Martello and Toth [20, 21] used the above dominance criterion in their MTRP reduction procedure. An iterative procedure to reduce the size of an instance of BP could enumerate all feasible sets, search for a feasible set F dominating all others, assign F to a new bin, and remove the items in F from N, until either no such F exists or all items have been placed. Since this is clearly impractical, the reduction procedure MTRP limits the search to feasible sets of cardinality at most three.

3. Lower and upper bounds

Let S be a (not necessarily feasible) solution to BP and denote by $z(S)$ its number of bins. Associated with solution S there is a family of subsets $B_1, \ldots, B_{z(S)}$ representing each bin, where B_j is used to denote both the j-th bin itself and the set of items it contains, for every $j = 1, \ldots, z(S)$. 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 $S, j = 1, \ldots, z(S)$. Then, each bin B_j is in exactly one of the following situations:

- *violated*: $w_S(j) > C$
- *full* or *complete*: $w_S(j) = C$
- *incomplete*: $C > w_S(j) > 0$.
- *empty*: $w_S(j) = 0$

A bin is *saturated* if it is violated or full. Bins which are not violated are said to be *feasible*.

The reader is referred to [5, 20] for reviews of greedy algorithms for BP. They usually start by sorting the items in non-increasing order of their weights. Next, items are picked in this order and placed one-at-a-time in a bin selected according with some strategy. In this work, we use the Best-Fit Decreasing (BFD) heuristic to compute feasible solutions to BP. This algorithm always selects the bin with the smallest sufficient residual capacity. If none of the bins already in use has enough capacity to accommodate the new item, then a new bin is opened. Algorithm BFD can be implemented to run in $O(n \log n)$ time.

Lower bounds to BP are reviewed by Scholl et al. [24]. We describe below the bounds used in this work. A trivial lower bound to BP is given by $L_1 = \lceil \sum_{i=1}^{n} w_i / C \rceil$. The dominating lower bound $L_2 \geq L_1$ originally described in [20] considers a partition of N into two subsets J_1 and J_2 defined by an integer parameter $0 \leq \alpha \leq C/2$, where $J_1 = \{j \in N : w_j > C - \alpha\}$ and $J_2 = \{j \in N : C - \alpha \geq w_j \geq \alpha\}$. Then, $L(\alpha) = |J_1| + \lceil (1/C) \sum_{j \in J_2} w_j \rceil)$ is a lower bound on the number of bins required for packing all items, and $L_2 = \max\{L(\alpha) : 0 \leq \alpha \leq C/2\}$. This bound can be computed in $O(n)$ time, once all items are sorted in non-increasing order of their weights.

A function $u : [0, 1] \to [0, 1]$ is called *dual feasible* if $\sum_{w \in S} w \leq 1 \Rightarrow \sum_{w \in S} u(w) \leq 1$ for any finite set S of nonnegative real numbers. This term refers to the fact that for any dual feasible function u and for any instance of the bin packing problem with item weights $w_1, \ldots, w_n$, then $(u(w_1), \ldots, u(w_n))$ is a feasible solution for the dual of the corresponding fractional bin packing problem [8, 18]. Fekete and Schepers [8] presented a simple and fast approach for obtaining a class of lower bounds based on dual feasible functions. The use of several dual feasible functions make it possible to improve the bounds obtained when a single function is used. They introduced a particular class of bounds $L_*^{(p)}$ that can be computed in time $O(n)$ for any $p \geq 2$, provided that all items are previously sorted in non-increasing order of their weights. The bound $L_*^{(p)}$ with $p = 20$ is used in this work.

The reduction procedure described in Section 2 was used by Martello and Toth [20, 21] to compute a lower bound L_3 as follows. Let I be the original problem instance, b_1 the number of bins reduced after the application of MTRP to I, and I_1 the corresponding residual instance. The application of lower bound L_2 to I_1 provides a lower bound $L'_1 = b_1 + L_2(I_1)$ to I. Next, I_1 is relaxed by the removal of the lightest item and MTRP is applied to this relaxed instance, yielding b_2 fixed bins and the new residual problem I_2. Thus, $L'_2 = b_1 + b_2 + L_2(I_2)$ is a lower bound to the original problem. This process is repeated until the residual instance is empty. A lower bound $L'_k = b_1 + b_2 + \ldots + b_k + L_2(I_k)$ is produced at each iteration k. Then, $L_3 = \max\{L'_1, L'_2, \ldots, L'_{k_{max}}\}$ is a valid lower bound to the original instance, where k_{max} is the iteration counter for which the residual instance becomes empty. This procedure to compute the bound L_3 can also be used to further improve the reduction obtained by the application of MTRP to the original instance. At any of its iterations, if all items previously removed can be assigned to the bins fixed by MTRP, then this reduction is also valid for the original problem. In particular, if all the removed items have been assigned at the end, then this solution is optimal. Since MTRP runs in $O(n^2)$ time, the complexity of the computation of L_3 is $O(n^3)$. The lower bound L_3 dominates L_2.

We propose in this work a new *destructive bound* [24], based on the work of Dell'Amico and Martello [3] for the multiprocessor scheduling problem (P$||$C$_{max}$) and on the yet unpublished work of Schoenfield [22]. 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

$$\theta = \max_{q=1,\ldots,n} \{q : \sum_{i=n-q+1}^{n} w_i \leq C\}$$

be an upper bound to the number of items in any bin of a feasible solution to BP.

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

Proof. We suppose the existence of a solution using m bins. Any feasible solution to BP has at most θ items in each bin. The average number of items by bin is n/m. Then, at least one bin contains $\lceil n/m \rceil$ items or more. Thus, if $\theta < \lceil n/m \rceil$ there is no feasible solution using m bins. □

Proposition 2. *Given an integer $\sigma \leq \lfloor n/m \rfloor$, 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.*

Proof. For any integer value of $\sigma \leq \lfloor n/m \rfloor$, we attempt to build a feasible solution with m bins in which any of them contains at most $\sigma - 1$ items. Without loss of generality, suppose the first bin contains $\sigma - 1$ or fewer items. Then, the remaining $n - (\sigma - 1)$ items should be packed in $m - 1$ bins. Let $C(\sigma) = \lceil \sum_{i=\sigma}^{n} \frac{w_i}{m-1} \rceil$ be a lower bound to the smallest bin capacity capable of packing the smallest items, indexed by $\sigma, \sigma+1, \ldots, n$, in $m-1$ bins. If $C(\sigma) > C$, then every bin will contain σ or more items. □

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

Proof. cf. [3, page 194, Proposition 4]. □

The next theorem follows directly from Propositions 2 and 3:

Theorem 1.

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

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.*

Proof. A feasible solution using m bins has at least $\underline{m}_\vartheta$ bins with exactly ϑ items in each of them. We now consider any solution in which the $\underline{m}_\vartheta\cdot\vartheta$ heaviest items are accommodated in the first $\underline{m}_\vartheta$ bins. If $\lceil\sum_{i=\underline{m}_\vartheta\cdot\vartheta+1}^{n} \frac{w_i}{(m-\underline{m}_\vartheta)}\rceil > C$, the remaining $n-\underline{m}_\vartheta\cdot\vartheta$ items cannot be accommodated in the remaining $m-\underline{m}_\vartheta$ bins and the lower bound to the number of bins in a feasible solution may be increased by one. □

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.*

Proof. We consider any solution with m_θ bins with exactly θ items in each of them and with m_ϑ bins with exactly ϑ items in each of them. If the bin capacity needed to accommodate the $m_\theta\cdot\theta$ lightest items in m_θ bins is larger than C or if the bin capacity needed to accommodate the $m_\vartheta\cdot\vartheta$ lightest items in m_ϑ bins is larger than C, then there is no feasible solution using m bins and the lower bound to the number of bins in a feasible solution may be increased by one. □

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,*

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

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

If $\theta > \vartheta+1$, the following reduction can still be used as an attempt to improve the current lower bound. If $\vartheta < \lceil n/m \rceil$, we attempt to show that no feasible solution may have a bin with exactly only ϑ items, in which case ϑ can be increased by one and the lower bound L_ϑ can be recomputed. To do so, we accommodate the heaviest ϑ items in a single bin. Next, we consider the reduced instance defined by th $n' = n - \vartheta$ items with weights $w_{\vartheta+1}, \ldots, w_n$. In this situation, $m' = m - 1$ is a lower bound to the number of bins in any feasible solution of this reduced instance. The bound $L_\vartheta(m')$ for this new instance can be computed following Theorem 2. If $L_\vartheta(m') > m'$, then there is no feasible solution for the original instance in which a bin has exactly ϑ items. If this is the case, then ϑ can be increased by one and the original bound L_ϑ can be recomputed.

To illustrate the computation of L_ϑ, we consider a bin packing instance with $n = 20$, $w = (54, 54, 53, 53, 53, 52, 51, 51, 51, 50, 50, 48, 48, 46, 46, 33, 32, 32, 32, 31)$, and $C = 120$. We first compute the lower bound $L_1 = \lceil 920/120 \rceil = 8$ and $\theta = 3$ (maximum number of items in a bin). Next, Theorem 1 is applied with $m = 8$ and we get $\vartheta = 2$ (minimum number of items in a bin). Since $\theta = \vartheta + 1$, we obtain $m_\vartheta = 4$ and $m_\theta = 4$. Then,

$$C_\vartheta(8) \quad = \max\left\{ \left\lceil \sum_{i=13}^{20} w_i/4 \right\rceil, \left\lceil \sum_{i=9}^{20} w_i/4 \right\rceil \right\} = 125 > C,$$

and, according with Proposition 5, $L_\vartheta = m + 1 = 9$ is a valid lower bound.

4. Initial solutions

Each iteration of our hybrid improvement heuristic starts by creating a feasible solution to DBP using a fixed number of bins, then attempts to transform it so as to find a feasible solution to BP using the same number of bins. We used three deterministic construction procedures for building feasible solutions to DBP. All of them start with LB open bins and investigate the items in non-increasing order of their weights. The following bin selection rules are applied in each case:

- Dual Best-Fit Decreasing (DBFD): Select the bin with smallest sufficient residual capacity; if none is available then select the lightest bin.
- Dual Subset 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 the polynomial-time approximation scheme MTSS(3) of Martello and Toth [20]). Otherwise, the current item is inserted into the lightest bin. DSSFD is similar to a variant of the Minimum Bin Slack heuristic described in [9], which also attempts to fill a bin after fixing the heaviest unselected item. It is also similar to a pseudopolynomial heuristic [28] based on multiple solutions of the subset sum problem, as well as to the procedure *Fill Bin* [31] and to the heuristic *Multi-Subset* [3].
- 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 nonsaturated bin in which it fits; if none is available then the lightest bin is selected.
- Longest Processing Time (LPT): This strategy is derived from the LPT scheduling heuristic [13]. The lightest bin is always selected.

5. Load redistribution

Whenever a feasible solution to DBP is not feasible to BP, load balancing and load unbalancing substrategies are applied to improve bin usability by load redistribution. Each substrategy is preceded by the application of an item dominance rule.

5.1. **Item dominance rule.** A local item dominance rule is always applied before the two first phases. We say that an item i_3 dominates two items i_1 and i_2 in another bin if $w_{i_3} = w_{i_1} + w_{i_2}$. This rule is applied as follows. For every *full* bin B_i and for every *incomplete* or *violated* bin B_j in the current solution, whenever there exist two items $i_1, i_2 \in B_i$ and one item $i_3 \in B_j$ such that $w_{i_1} + w_{i_2} = w_{i_3}$, then items i_1 and i_2 are exchanged with i_3 in the current solution. Although the weights of both bins involved in the exchange remain unchanged, increasing the number of small items in the incomplete or violated bins will give more chance to the load balancing and unbalancing phases to find a better solution.

The search for dominant items stops when no further improvement is possible, after all possible pairs of bins have been evaluated. It does not change the weights of the bins, but may help in reaching feasibility at a later step.

5.2. **Load balancing by differencing.** Given any pair of bins B_i and B_j 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 bins. The differencing method of Karmarkar and Karp [17] begins by arranging the items in these bins into a non-increasing ordered list. The method recursively takes the difference of the two greatest values remaining in the ordered list and places this difference back into the ordered list, as if it constituted a virtual item to be packed. The differencing continues until only one value remains in the list. This remaining value represents the difference of an implied partition of the original list into two bins. The new partition is then constructed by backtracking through the recursion. This substrategy continually seeks to knock down excess deviations, starting by the largest capacity violation.

The above algorithm and its randomized version [4] are applied once each to all pairs of bins of the current solution, in which one of them is *violated* and the other is *non-saturated*. The first pair is that formed by the lightest and the heaviest bins. The violated bins are investigated in non-increasing order of their weights, while the nonsaturated ones are investigated in the opposite order. For each pair of bins B_i and B_j, the best solution S' among the two newly computed partitions replaces the current one if $|w_{S'}(i) - w_{S'}(j)| < |w_S(i) - w_S(j)|$. The search stops when no further improvement is possible.

5.3. **Load unbalancing.** Given a set $N = \{1, \ldots, n\}$ of items with weights $w_i, i = 1, \ldots, n$, 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 substrategy 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 of Martello and Toth [20] with $k = 3$. 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. We give in Figure 1 a short description of the procedure that combines the item dominance rule with the load balancing and unbalancing substrategies, which is applied to the infeasible solution S built at the construction phase.

procedure `Redistribution`(S);
1 Apply the load balancing substrategy by differencing preceeded by the item dominance rule;
2 **if** a feasible solution to BP was obtained **then return** S;
3 Apply the load unbalancing substrategy preceeded by the item dominance rule;
4 **if** the current solution was not changed in step 3 **then return S**;
5 Go back to step 1;
end `Redistribution`.

FIGURE 1. Pseudo-code of the load redistribution phase

6. INFEASIBILITY REDUCTION

We apply a tabu search strategy to reduce capacity violations in the current solution. For any solution S, we denote by $E_S = \sum_{j=1}^{z(S)} \max\{0, w_S(j) - C\}$ 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 $S'(i) = S(k)$, $S'(k) = S(i)$, $S'(\ell) = S(\ell)\ \forall \ell \neq i, k$.

Since the ultimate goal of the search is to make an infeasible solution feasible, every violated bin has to be made feasible. Therefore, we only consider moves that decrease the excess deviation of the target violated bin. By this filtering process, each iteration will consider exclusively 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.

Table 1 shows all possible situations for a move $i \leftrightarrow k$ involving bins $S(i)$ and $S(k)$: the move type, the status of each bin after the move, the associated move value $\Delta(i,k)$, the number of violated bins in the pair after the move, the number of complete bins in the pair after the move, and an associated level value. 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 move type by a "level" or "priority". For a given target violated bin whose neighborhood is being investigated, one chooses the move with the smaller level (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 move types (since one move type may be better than another with respect to one criterion, but not with respect to the other), two different potential level values are assigned to some move types and one of them is randomly selected with probability 1/2 at each iteration. Such 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.

Type	Status of the bins after move $i \leftrightarrow k$	$\Delta(i,k)$	Violated	Complete	Level
1	*complete, complete*	0	0	2	1
2	*complete, incomplete*	0	0	1	2
3	*incomplete, incomplete*	0	0	0	3
4	*complete, violated*	>0	1	1	4,2
5	*incomplete, violated*	>0	1	0	5,4
6	*violated, violated*	>0	2	0	6,5

TABLE 1. Move types

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 $S(i)$. In our implementation, the value of `TabuTenure` is randomly chosen from a discrete uniform distribution in the interval $[0.8 \cdot \sqrt{n}, 1.2 \cdot \sqrt{n}]$.

Tabu search proposes the use of logical restructuring based on anticipatory analysis. In this context, logical restructuring seeks to answer the following questions: "What conditions assure the existence of a trajectory that will lead to an improved solution?" and "What intermediate moves can create such conditions?" Intermediate moves may be generated either by modifying the evaluations used to select transitions between solutions or by modifying the neighborhood structure that determines these transitions [12]. Moves of type 1 are the most attractive, since they result in two complete bins. Although moves of types 4, 5, and 6 are bad in principle, they may be effective in some situations. This will be the case, for example, if there is a move of type 1 associated with one of the resulting violated bins. Then, performing these two subsequent moves at once as a combined move will generate a new solution with more complete bins. To implement this idea, we store the temporary solution $\overline{S}$ generated by each move of type 4, 5, and 6. Let $B_{\overline{j}}$ be the violated bin in case the move type is 4 or 5, otherwise let $B_{\overline{j}}$ be the current target bin. If there exists a move of type 1 involving an item from bin $B_{\overline{j}}$ of the temporary solution $\overline{S}$, then the two combined moves are performed.

We summarize in Figure 2 the procedure `MoveSelect`, which computes and returns the best move to be applied to a target violated bin B_j of the current solution S. We denote by $S' = S(i,k)$ the solution derived from S by swapping items i and k from the bins where they are currently placed. `Feasible`(S') returns `.TRUE.` if S' is feasible, `.FALSE.` otherwise. `Tabu`(i,k) returns `.TRUE.` if the swap move $i \leftrightarrow k$ is forbidden, `.FALSE.` otherwise. Variables associated with the best move are initialized in lines 1-2. The nested loops in lines 3-27 and 4-26 enumerate all candidate moves, defined by pairs formed by each item in bin B_j and every item with a smaller weight in any other bin. Line 5 determines if move $i \leftrightarrow k$ is of type 1 or leads to a feasible solution, in which case it is immediately selected and returned in lines 7-8. Otherwise, we check in line 10 if the move type is either 4, 5, or 6, and if the total capacity violation $\Delta(i,k)$ of bins $S(i)$ and $S(k)$ in the new solution is less than a certain threshold `MaxDelta`. In this case, we search in lines 11-23 if there exists a move of type 1 to be combined with $i \leftrightarrow k$. A temporary solution $\overline{S}$ derived from the current solution S by the application of move $i \leftrightarrow k$ is computed in line 12. The target bin $B_{\overline{j}}$ for the next move is determined in line 13. The nested loops in lines 14-22 and 15-21 play the same roles as those in lines 3-27 and 4-26, generating all moves involving an item $\overline{i}$ from bin $B_{\overline{j}}$ and every item $\overline{k}$ from another bin. Line 16 determines if move $\overline{i} \leftrightarrow \overline{k}$ applied to the temporary solution $\overline{S}$ is of type 1 or leads to a feasible solution, in which case it is immediately selected and returned in lines 18-19 together with the first move. Line 24 handles the unforbidden moves which do not fall in the previous cases. If move $i \leftrightarrow k$ improves the best move already found, then all information about the incumbent is updated in line 25. In case neither a feasible solution nor a type 1 move has been identified by the selection procedure, the best move or an indication that no unforbidden move exists is returned in line 28.

The overall tabu search procedure for infeasibility reduction starting from the current solution S is summarized in Figure 3. Initializations are performed in lines 1-2. The bins are sorted in non-increasing order of their weights in line 3. The target bin is initialized in line 4. Iterations are performed along the loop in lines 5-29 until a feasible solution is found or some stopping criterion is attained. At each iteration, the algorithm determines in line 6 the best move (single or combined) associated with the target bin B_j (if one exists). This move is applied to the current solution in lines 7-20. If all moves associated with the target bin were forbidden or if the latter was made feasible after the selected move, then the target bin is incremented by one in line 21. Line 22 detects if all violated bins have been investigated and the current solution is still infeasible, in which case the bins are reordered according to their current weights and the target bin is reset to 1 in lines 23-26. The target bin and the number of tabu search iterations are updated in lines 27-28 and a new iteration starts. The infeasibility reduction phase stops if a feasible solution is found or if a total of `MaxIterations` tabu search iterations have been performed without improvement in the total excess violation.

```
procedure MoveSelect(S, j);
BestLevel, BestValue ← ∞;
moves ← 0; BestMove, BestMove2 ← ∅;
forall i ∈ B_j do
    forall k ∈ {1, ..., n} \ B_j : w_k < w_i do
        if Feasible(S(i, k)) or MoveType(i ↔ k) = 1
        then do
            BestMove ← i ↔ k; moves ← 1;
            return moves, BestMove, BestMove2;
        end;
        if (MoveType(i ↔ k) = 4 or MoveType(i ↔ k) = 5 or
            MoveType(i ↔ k) = 6) and Δ(i, k) < MaxDelta
        then do
            S̄ ← S(i, k);
            if w_S̄(S̄(i)) > C then j̄ ← S̄(i) else j̄ ← S̄(k);
            forall ī ∈ B_j̄ do
                forall k̄ ∈ {1, ..., n} \ B_j̄ : w_k̄ < w_ī do
                    if Feasible(S̄(ī, k̄) or MoveType(ī ↔ k̄) = 1
                    then do
                        BestMove ← i ↔ k; BestMove2 ← ī ↔ k̄; moves ← 2;
                        return moves, BestMove, BestMove2;
                    end;
                end;
            end;
        end;
        if .NOT.Tabu(i ↔ k) and (MoveLevel(i ↔ k) < BestLevel or
            (MoveLevel(i ↔ k) = BestLevel and Δ(i, k) < BestValue))
        then do BestMove ← i ↔ k; BestValue ← Δ(i, k); moves ← 1; end;
    end;
end;
return moves, BestMove, BestMove2;
end MoveSelect.
```

FIGURE 2. Move selection procedure

7. THE HYBRID IMPROVEMENT HEURISTIC

The full hybrid solution improvement heuristic HI_BP for the bin packing problem is outlined in Figure 4, integrating all previously described phases. The preprocessing phase is performed in lines 1-11. A feasible solution S_{BFD} to BP is computed in line 1 by the BFD greedy heuristic. The upper bound UB to the number of bins is set in line 2 as the number of bins in S_{BFD}. The bounds L_1 and L_2 are computed respectively in lines 3 and 4. If any of them is equal to the upper bound UB, then the optimal solution S_{BFD} is returned. The lower bound L_3 is computed in line 5 using the reduction procedure MTRP, which builds the partial solution S_{MTRP} with b bins. If the lower bound L_3 is equal to the number of bins in S_{MTRP}, then the original instance was completely reduced and the optimal slution S_{MTRP} is

procedure `TabuSearch`(S);
`iterations` $\leftarrow 0$;
`TabuList` $\leftarrow \emptyset$;
Sort the bins in non-increasing order of their weights, obtaining a sequence of indices $i_1, \ldots, i_{z(S)}$ such that $w_S(i_k) \geq w_S(i_{k+1}), k = 1, \ldots, z(S) - 1$;
$current \leftarrow 1$; $j \leftarrow i_{current}$;
while stopping criterion is not matched **and** S is infeasible **do**
  (`moves`, $i \leftrightarrow k, \bar{i} \leftrightarrow \bar{k}$) $\leftarrow$ `MoveSelect`(S, j);
  if `moves` ≥ 1
  then do
    Compute `TabuTenure` $\in U[0.8 \cdot \sqrt{n}, 1.2 \cdot \sqrt{n}]$;
    Forbid the reinsertion of item i into bin $S(i)$ for the next `TabuTenure` iterations;
    Forbid the reinsertion of item k into bin $S(k)$ for the next `TabuTenure` iterations;
    Update the current solution S: $a \leftarrow S(i)$, $b \leftarrow S(k)$, $S(i) \leftarrow b$, and $S(k) \leftarrow a$;
    if `moves` $= 2$
    then do
      Compute `TabuTenure` $\in U[0.8 \cdot \sqrt{n}, 1.2 \cdot \sqrt{n}]$;
      Forbid the reinsertion of item $\bar{i}$ into bin $S(\bar{i})$ for the next `TabuTenure` iterations;
      Forbid the reinsertion of item $\bar{k}$ into bin $S(\bar{k})$ for the next `TabuTenure` iterations;
      Update the current solution S: $a \leftarrow S(\bar{i})$, $b \leftarrow S(\bar{k})$, $S(\bar{i}) \leftarrow b$, and $S(\bar{k}) \leftarrow a$;
    end;
  end;
  if $w_S(j) \leq C$ **or** `moves` $= 0$ **then** $current \leftarrow current + 1$;
  if $current > z(S)$ **or** ($current \leq z(S)$ **and** $w_S(i_{current}) \leq C$)
  then do
    Sort the bins in non-increasing order of their weights, obtaining a new sequence of indices $i_1, \ldots, i_{z(S)}$ such that $w_S(i_k) \geq w_S(i_{k+1}), k = 1, \ldots, z(S) - 1$;
    $current \leftarrow 1$;
  end;
  $j \leftarrow i_{current}$;
  `iterations` $\leftarrow$ `iterations` $+ 1$;
end;
return S;
end `TabuSearch`.

FIGURE 3. Tabu search procedure for infeasibility reduction

returned in line 6. A new test involving the lower bound L_3 is performed in line 7: if the latter is equal to the upper bound UB, then the optimal solution S_{BFD} is returned. The items accommodated in the partial solution S_{MTRP} are removed from the set N of items and the bounds L_3 and UB to the reduced instance are updated in line 9. In line 10, the lower bound LB and the tentative number of bins $nbins$ to be used are set as the best among bounds L_3, $L_*^{(20)}$, and L_ϑ (see Section 3). If the newly recomputed lower bound allows the identification of an optimal solution, then S_{BFD} is returned in line 11.

The next steps correspond to the improvement heuristic itself. Variable $infeasible$ used for the implementation of the stopping criterion is initialized in line 12. The loop in lines 13-32 searches for a feasible solution to BP using $nbins$ bins. The search stops if a feasible solution with $nbins$ bins is found or if the lower bound reaches the upper bound. The loop in lines 14-30 performs at most `MaxTrials` attempts to generate a feasible solution using $nbins$ bins. Each attempt starts in line 15 by the selection of a still unused construction algorithm from the set $H = \{$DB3FD, DBFD, DSSFD, LPT$\}$ of available heuristics to DBP. As many attempts as the number of heuristics in H are performed. The selected heuristic is used to build a feasible solution S to DBP in line 16. The upper bound and the feasibility flag are updated in lines 17-18 if this solution is feasible to BP. Otherwise, the load redistribution procedure described in Section 5 and depicted in Figure 1 is applied in line 20. If the new solution is feasible to BP, the upper bound and the feasibility flag are updated in lines 21-22. Otherwise, the tabu search procedure for infeasibility reduction described in Section 6 and depicted in Figure 3 is applied in line 24. If the new solution is feasible to BP, the upper bound and the feasibility flag are updated in lines 25-26. The tentative number of bins $nbins$ is incremented by one in line 31 if the algorithm failed to generate a feasible solution to BP after `MaxTrials` attempts. If the loop in lines 13-32 found a better solution to BP, then the solution corresponding to the bins in S and those in the partial solution S_{MTRP} is returned in line 33, otherwise the initial solution S_{BFD} is returned in line 34.

The following parameter settings have been used in our implementation:

- `MaxTrials` $= 4$ (number of attempts made by the solution improvement heuristic `HI_BP` to build a feasible solution to BP using a lower bound LB)
- `MaxDelta` $= 0.2 \times \min_{i=1,\dots,n} w(i)$ (threshold used by the selection procedure `MoveSelect` to determine whether a combined move will be searched for or not)
- `MaxIterations` $= 4000$ (maximum number of iterations without improvement in the excess violation E_S, used as a stopping criterion by the tabu search procedure).

8. Computational experiments

We report in this section the computational experiments performed with the hybrid improvement heuristic on a broad set of test problems. Heuristic `HI_BP` was coded in `C` and compiled with version 2.95.2 of `gcc`, using the optimization flag -O3. The reduction procedure MTRP and the computation of the lower bound L_3 were also implemented in `C`, following as closely as possible the original Fortran code in [20].

All experiments were performed on a 1.7 GHz Pentium IV with 256 Mbytes of RAM memory. The computation times reported in this section are given in seconds. Moreover, given a proven optimal solution value (or the best lower bound in case the latter is not available) x and the solution value (i.e., the number of bins) y

```
procedure HI_BP
Compute the solution $S_{BFD} = \{B_1, \dots, B_{z(S_{BFD})}\}$ using algorithm BFD;
Set $UB \leftarrow z(S_{BFD})$;
if $L_1 = UB$ then return $S_{BFD}$;
if $L_2 = UB$ then return $S_{BFD}$;
Compute the lower bound $L_3$ and let $S_{MTRP} = (B_1, \dots, B_b)$ be the partial
    solution with $b$ bins created by the reduction procedure MTRP;
if $L_3 = b$ then return $S_{MTRP}$;
if $L_3 = UB$ then return $S_{BFD}$;
Remove from $N$ the items accommodated in the partial solution $S_{MTRP}$;
Update the bounds $L_3 \leftarrow L_3 - b$ and $UB \leftarrow UB - b$;
Compute the lower bounds $L_*^{(20)}$ and $L_\vartheta$ and set $LB, nbins \leftarrow \max\{L_3, L_*^{(20)}, L_\vartheta\}$;
if $nbins = UB$ then do $S \leftarrow S_{BFD}$; return $S$; end;
$infeasible \leftarrow$ .TRUE.;
while $nbins < UB$ do
    for $k = 1, \dots,$ MaxTrials and $infeasible =$ .TRUE. do
        Select a heuristic from $H = \{$DB3FD, DBFD, DSSFD, LPT$\}$;
        Build a solution $S = \{B_1, \dots, B_{z(S)}\}$ to DBP;
        if $S$ is feasible to BP then do
            Set $infeasible \leftarrow$ .FALSE. and $UB \leftarrow z(S)$;
        else do
            $S \leftarrow$ Redistribution($S$)
            if $S$ is feasible to BP then do
                Set $infeasible \leftarrow$ .FALSE. and $UB \leftarrow z(S)$;
            else do
                $S \leftarrow$ TabuSearch($S$)
                if $S$ is feasible to BP then do
                    Set $infeasible \leftarrow$ .FALSE. and $UB \leftarrow z(S)$;
                end;
            end;
        end;
    end;
    if $infeasible =$ .TRUE. then set $nbins \leftarrow nbins + 1$;
end;
if .NOT.$infeasible$ then return $S \cup S_{MTRP}$;
else return $S_{BFD}$;
end HI_BP.
```

FIGURE 4. Pseudo-code of the hybrid improvement procedure for the bin packing problem

obtained by heuristic HI_BP, we define their *absolute difference* as $y - x$ and their *relative diference* as $(y - x)/x$.

8.1. **Test problems.** We first considered two classes of test problems introduced by Falkenauer [7]:

- `uniform`: formed by 80 instances with bin capacity $C = 150$ and randomly generated weights between 20 and 100. There are 20 instances for each value of $n = 120$ (`u_120`), $n = 250$ (`u_250`), $n = 500$ (`u_500`), and $n = 1000$ (`u_1000`). The optimal solutions of all these instances are known, see Valério de Carvalho [30] and Gent [11]. Their optimal values coincide with the simple lower bound L_1 for 79 instances.
- `triplets`: harder problems in which each bin of the optimal solution is completely filled with exactly three items. Their optimal values also coincide with the lower bound L_1. Item weights are drawn from the range (250,500) with bins of capacity $C = 1000$. There are 20 instances for each value of $n = 60$ (`t_60`), $n = 120$ (`t_120`), $n = 249$ (`t_249`), and $n = 501$ (`t_501`).

We also used three classes of test problems from Scholl et al. [24]:

- `set_1`: formed by 720 instances with $n = 50, 100, 200, 500$, bin capacity $C = 100, 120, 150$, and weights uniformly generated from different intervals $[1, 100]$, $[20, 100]$, and $[30, 100]$, constructed in a similar way to some of those proposed by Martello and Toth [20].
- `set_2`: formed by 480 instances with $n = 50, 100, 200, 500$ and bin capacity $C = 1000$, generated to accommodate more items (three to nine items per bin in the average) in their optimal solutions than those in `set_1`.
- `set_3`: formed by ten difficult instances with $n = 200$, bin capacity $C = 100000$, and item weights drawn from the range $[20000, 35000]$.

Reference [23] reports 704 proven optimal values for the instances of `set_1`, 477 for the instances of `set_2`, and three for the instances of `set_3`. The optimal values of the 26 open instances in [23] were identified by using the new lower bound L_ϑ proposed in Section 3 or the cutting stock lower bound L_{CS} (previously used by Schwerin and Wäscher [27]). With the exception of the four last instances of set `set_1`, all optimal values coincide with the reported upper bounds.

Three other test sets were also used:

- `was_1`: 100 instances from file `sch_wae1.bpp` with bin capacity $C = 1000$, $n = 100$, minimum item weight equal to 150, and maximum item weight equal to 200.
- `was_2`: 100 instances from file `sch_wae2.bpp` with bin capacity $C = 1000$, $n = 120$, minimum item weight equal to 150, and maximum item weight equal to 200.
- `gau_1`: 17 instances from file `wae_gau1.bpp`

Instances from sets `was_1` and `was_2` were collected from [26, 27]. Set `gau_1` is formed by residual problems collected from [32] and reported by their authors as difficult. All instances in these sets are available from [29].

Schwerin and Wäscher [26] described a problem generator and new classes of test instances, characterized by four parameters: the bin capacity C, the number of items n, a lower bound $v_1 \cdot C$ to the item weights, and an upper bound $v_2 \cdot C$ to the item weights. Different combinations of these parameter values lead to 440 different classes of test problems. They generated 100 instances in each class. For a given class, let p denote the number of instances from this class which were solved to optimality by heuristic FFD. The authors defined that a class is said to be `extremely-ffd-hard` if $p < 20$, `ffd-hard` if $20 \leq p < 80$, or `ffd-easy` if

$p \geq 80$. We used the instance generator BPPGEN available in [29] to create 100 hard instances (i.e., instances for which the value of the solution obtained by FFD is different from L_{CS}) of each of the 145 `ffd-hard` or `extremely-ffd-hard` classes and used them in the final part of our computational experiments, as reported in Section 8.3.

8.2. **Phases of `HI_BP`.** We now investigate the effectiveness of the different phases of `HI_BP`. To do so, we created three different variants of our heuristic. Variant `C` performs only the construction phase. Variant `C+R` is an extension of the latter, which also performs the redistribution phase. Finally, variant `C+R+I` corresponds to the full algorithm `HI_BP` and includes all phases (construction, redistribution, improvement). Each variant may perform up to four attempts to build a feasible solution to BP using the same lower bound. For this experiment, we consider the subset formed by the 581 instances from sets `uniform`, `triplets`, `set_1`, `set_2`, and `set_3`, for which the solution found by `HI_BP` was not found directly by algorithm BFD (line 2) or by MTRP (line 1).

The main results are summarized in Table 2. For each class, we first indicate the number of instances. Next, we give the results observed by each variant: the number of optimal solutions found and the total computation time in seconds over all instances. We also give the percentage of the total time taken exclusively by the last phase (improvement). The construction phase alone is very weak and the optimal solutions for many instances are missed. Only 58% of the latter are solved to optimality. The redistribution phase contributed to significantly increase the number of instances solved to optimality, as well as to reduce the computation times (since fewer attempts to find a feasible solution are performed). However, no optimal solution was found for any of the particularly difficult triplet instances. The two first phases combined were able to find the optimal solutions for 472 out of the remaining (i.e., non-triplet) 501 instances (94%). The final improvement phase is essential to solve the harder `triplets` class, whose optimal solutions have all their bins completely full. The execution of the improvement phase improved upon both the number of optimal solutions found and the computation times. Together, the three phases of heuristic `HI_BP` have been able to found the optimal solutions to all 581 test instances in this set.

		`C`		`C+R`		`C+R+I`		
Class	inst.	opt.	time (s)	opt.	time (s)	opt.	time (s)	time (%)
`uniform`	74	41	42.30	68	1.92	74	2.65	87.17
`triplets`	80	0	144.97	0	123.39	80	78.36	99.06
`set_1`	173	111	5.28	154	2.94	173	122.00	98.25
`set_2`	244	185	171.99	240	44.56	244	6.09	71.59
`set_3`	10	0	4.39	10	1.73	10	46.00	96.48
Total	581	337		472		581		

TABLE 2. Results after each phase of the hybrid improvement heuristic

8.3. **Comparison with other approaches.** We now compare the results obtained by the hybrid improvement procedure with those obtained by other approaches reported in the literature. Heuristic `HI_BP` was applied only once for each instance, using the initial seed set to 1.

HI_BP compares favorably with other approaches. We solved to optimality all instances in the uniform and triplets classes solved by the exact method proposed by Valério de Carvalho [30]. In the following, we compare our results with those obtained by BISON [24] and by the best strategy Perturbation MBS' + VNS among those recently proposed by Fleszar and Hindi [9], considering the same classes of instances used in these references. We first report in Table 3 the results obtained by HI_BP and Perturbation MBS' + VNS. For each class, we give the number of instances and, for each heuristic, the number of instances for which the optimal solution was found, the maximum absolute deviation, the maximum relative deviation, and the average and maximum computation times in seconds. The results reported for Perturbation MBS' + VNS were obtained on a 400 MHz Pentium and extracted from Table VIII of [9]. Similar statistics are reported in Table 4, concerning the comparison between HI_BP and BISON. The results obtained for BISON, with the processing time limited at 1000 seconds, were obtained on a PC 80486 DX2-66 and extracted from Tables 4, 5, and 6 of [24].

HI_BP found optimal solutions for all instances of sets set_1, set_2, and set_3, including the four instances from set_1 for which Alvim et al. [2] and Fleszar and Hindi [9] improved the best known solutions. HI_BP found optimal solutions for 41 additional instances with respect to Perturbation MBS' + VNS and for 37 additional instances with respect to BISON.

		HI_BP					Perturbation MBS' + VNS				
			max	max	time			max	max	time	
Class	instances	opt.	abs.	rel. (%)	avg	max	opt.	abs.	rel. (%)	avg	max
u_120	20	20	0	0	0.00	0.01	20	0	0.00	0.02	0.04
u_250	20	20	0	0	0.12	2.20	19	1	0.95	0.03	0.16
u_500	20	20	0	0	0.00	0.01	20	0	0.00	0.04	0.14
u_1000	20	20	0	0	0.01	0.02	20	0	0.00	0.07	0.27
t_60	20	20	0	0	0.37	1.65	20	0	0.00	0.01	0.01
t_120	20	20	0	0	0.85	4.53	20	0	0.00	0.02	0.04
t_249	20	20	0	0	0.22	0.51	20	0	0.00	0.02	0.04
t_501	20	20	0	0	2.49	19.26	20	0	0.00	0.06	0.10
set_1	720	720	0	0	0.19	19.23	694	2	2.44	0.15	1.78
set_2	480	480	0	0	0.01	1.19	474	1	2.94	0.10	4.57
set_3	10	10	0	0	4.60	44.76	2	1	1.85	3.74	5.05
Total	1370	1370					1329				

TABLE 3. HI_BP vs. Perturbation MBS' + VNS [9]

We also compared the results found by HI_BP with those reported in [29], obtained by heuristic MTPCS [27] with the backtracking limit set at 500, for the instances of sets was_1, was_2, and gau_1. Table 5 reports the solution values found by HI_BP for 11 instances for which it improved the best results reported in [29]. HI_BP missed the optimal solution for only five instances from gau_1 out of the 217 instances from was_1, was_2, and gau_1.

Finally, we compared the results obtained by HI_BP for the 145 ffd-hard and extremely-ffd-hard classes with those obtained by procedure MTPCS for the same classes, with the time limit set at 1000 seconds, extracted from Table 1 of [27].

		HI_BP					BISON			
			max	max	time			max	max	time
Class	instances	opt.	abs.	rel. (%)	avg	max	opt.	abs.	rel. (%)	avg
set_1	720	720	0	0	0.19	19.23	697	2	2.38	32.4
set_2	480	480	0	0	0.01	1.19	473	1	2.94	16.3
set_3	10	10	0	0	4.60	44.76	3	1	1.85	700.2
Total	1210	1210					1173			

Table 4. HI_BP vs. BISON [24]

Class	instance	new value	time
was_1	BPP81	18	0.07
was_2	BPP56	21	0.16
	BPP71	21	0.26
gau_1	TEST0097	12	0.01
	TEST0055	15	0.00
	TEST0049	11	0.00
	TEST0075	13	0.01
	TEST0054	14	0.00
	TEST0044	14	0.01
	TEST0095	16	0.01
	TEST0055	20	0.01

Table 5. Improved solutions for instances in sets was_1, was_2, and gau_1

Tables 6, 7, 8, and 9 report the number of optimal solutions obtained by MTPCS and the number of optimal solutions and the processing time (in seconds) of HI_BP for the 145 classes cited above. There are 100 instances in each of these classes, which are characterized by different values of v_1, v_2, and n (the value of $C = 1000$ is fixed). HI_BP obtained better results (i.e., more optimal solutions) than MTPCS for the instances in 95 out of the 145 classes: all 15 classes in Table 6; 33 out of the 37 classes in Table 7; 38 out of the 58 classes in Table 8; and 9 out of the 35 classses in Table 9. The superiority of HI_BP over MTPCS is striking for some of these hard classes of test problems. Considering e.g. the class defined by $v_1 = 0.15$, $v_2 = 0.5$, and $n = 200$ in Table 8, HI_BP found the optimal solutions for all 100 instances, while MTPCS obtained only one of them. MTPCS performed slightly better than HI_BP for only four out of the 145 classes of instances in these tables. Table 10 summarizes these results. HI_BP performed consistently better than MTPCS, solving to optimality 97.9% of the instances in the 145 classes ffd-hard and extremely-ffd-hard, while the latter found the optimal solutions for only 84.2% of the instances. The largest average processing time of HI_BP over all 145 classes was 5.54 seconds.

8.4. **Robustness.** We first give in Table 11 some statistics obtained over five runs of each instance from u_120, u_250, u_500, u_1000, t_60, _120, t_249, t_501, set_1, set_2, set_3, gau_1, was_1, and was_2 (7935 runs), with five consecutive seeds starting from 1. For each class, we first report the number of instances, the total number of runs, the number of runs for which the optimal solution was found,

v_1	v_2		n = 80	100	120	140	160	180	200
0.001	0.5	`MTPCS`					93		
		`HI_BP`					100		
		time (s)					0.01		
	0.6	`MTPCS`	87		79	75	83	41	
		`HI_BP`	95		96	99	99	99	
		time (s)	0.08		0.08	0.02	0.04	0.03	
	0.7	`MTPCS`		78	70	74	71	51	53
		`HI_BP`		91	88	90	92	92	91
		time (s)		0.16	0.23	0.25	0.22	0.24	0.29
	0.8	`MTPCS`				50		58	49
		`HI_BP`				80		84	79
		time (s)				0.74		0.76	0.93

TABLE 6. `HI_BP` vs. `MTPCS` [27] ($v_1 = 0.001$)

v_1	v_2		n = 40	60	80	100	120	140	160	180	200
0.05	0.1	`MTPCS`			100		99		99		99
		`HI_BP`			100		100		100		100
		time (s)			0.00		0.00		0.00		0.00
	0.2	`MTPCS`								100	
		`HI_BP`								100	
		time (s)								0.00	
	0.3	`MTPCS`								99	96
		`HI_BP`								100	100
		time (s)								0.00	0.00
	0.4	`MTPCS`								93	88
		`HI_BP`								100	100
		time (s)								0.00	0.00
	0.5	`MTPCS`	97			71	77	81	73	44	31
		`HI_BP`	99			100	100	99	100	100	100
		time (s)	0.01			0.00	0.00	0.03	0.01	0.00	0.00
	0.6	`MTPCS`	99	91	86	82	75	80	70	75	61
		`HI_BP`	99	98	95	98	98	98	100	99	99
		time (s)	0.03	0.04	0.07	0.04	0.04	0.05	0.03	0.04	0.04
	0.7	`MTPCS`		97		84	67	70	54	31	59
		`HI_BP`		95		92	91	94	88	84	94
		time (s)		0.08		0.19	0.21	0.19	0.37	0.50	0.25
	0.8	`MTPCS`					85	68	65	52	55
		`HI_BP`					94	86	89	91	88
		time (s)					0.47	0.63	0.79	0.74	0.89

TABLE 7. `HI_BP` vs. `MTPCS` [27] ($v_1 = 0.05$)

the maximum absolute deviation from the optimal value, and the average and the maximum computation times in seconds.

Heuristic `HI_BP` may find an optimal solution at four different points. We also report in this table the number of runs for which the optimal solution was found by preprocessing, reductions, or algorithm BFD (P0), at some time along the construction phase (P1), at some time along the redistribution phase (P2), or during the improvement phase (P3). As already observed by Scholl et al. [24], many instances from classes `set_1` and `set_2` are easily solved by phase (P0) alone. Once again, we notice that the tabu search algorithm used for infeasibility reduction in the improvement phase (P3) is absolutely necessary and was the only strategy which led to the optimal solutions of the `triplets` class.

The four last columns in Table 11 indicate the number of runs for which the optimal solution originated from an initial solution constructed by `DB3FD` (H1), `DBFD` (H2), `DSSFD` (H3), or `LPT` (H4), indicating the importance of using different strategies to build the initial solutions. We performed some preliminary computational

			n									
v_1	v_2		20	40	60	80	100	120	140	160	180	200
0.15	0.2	MTPCS		100	100	100	100	100	100	100	100	100
		HI_BP		100	100	100	100	100	100	100	100	100
		time (s)		0.01	0.00	0.02	0.02	0.15	0.04	0.09	0.08	0.07
	0.3	MTPCS		100	100	97	94	87	82	63	27	15
		HI_BP		100	100	100	100	100	100	100	100	100
		time (s)		0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	0.4	MTPCS		100	99	100	98	97	96	95	90	90
		HI_BP		100	100	100	100	100	100	100	100	100
		time (s)		0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01
	0.5	MTPCS	100	100	98	91	90	74	57	33	9	1
		HI_BP	99	100	99	100	100	100	100	100	100	100
		time (s)	0.00	0.01	0.03	0.00	0.05	0.08	0.08	0.03	0.04	0.07
	0.6	MTPCS		100	100	98	93	83	79	78	72	74
		HI_BP		100	98	100	98	100	97	98	96	100
		time (s)		0.01	0.03	0.05	0.05	0.09	0.11	0.17	0.27	0.15
	0.7	MTPCS		100	100	98	96	90	75	82	69	68
		HI_BP		100	100	98	99	96	100	91	96	92
		time (s)		0.00	0.03	0.11	0.15	0.44	0.40	1.24	0.95	1.63
	0.8	MTPCS								98	96	98
		HI_BP								99	98	96
		time (s)								0.28	0.50	0.74

TABLE 8. HI_BP vs. MTPCS [27] ($v_1 = 0.15$)

			n									
v_1	v_2		20	40	60	80	100	120	140	160	180	200
0.25	0.4	MTPCS	100	100	100	100	100	100	100	99	99	97
		HI_BP	100	100	100	100	100	100	100	100	100	100
		time (s)	0.00	0.00	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.00
	0.5	MTPCS	100	100	100	100	100	100	99	98	89	59
		HI_BP	100	100	100	100	100	100	100	100	100	100
		time (s)	0.00	0.00	0.06	0.05	0.10	0.50	1.14	2.44	2.77	5.54
	0.6	MTPCS		100	100	100	100	100	100	99	100	100
		HI_BP		100	100	100	100	100	100	100	100	100
		time (s)		0.00	0.01	0.03	0.06	0.05	0.09	0.13	0.14	0.17
	0.7	MTPCS			100			100	74	100	100	100
		HI_BP			100			100	100	100	100	100
		time (s)			0.00			0.05	0.04	0.12	0.12	0.10

TABLE 9. HI_BP vs. MTPS [27] ($v_1 = 0.25$)

			MTPCS		HI_BP	
classes	v_1	instances	opt.	%	opt.	%
extremely-ffd-hard	0.250	1900	1839	96.8	1900	100.0
	0.150	3600	2759	76.6	3567	99.1
ffd-hard	0.250	1600	1574	98.4	1600	100.0
	0.150	2200	2171	98.7	2183	99.2
	0.050	3700	2853	77.1	3568	96.4
	0.001	1500	1012	67.5	1375	91.7
Total		14500	12208	84.2	14193	97.9

TABLE 10. HI_BP vs. MTPCS [27]

experiments using each time only a randomized version of each of these four heuristics. We noted that even if more than Maxtrials $= 4$ attempts were allowed, none of them was able to build initial solutions capable of leading to optimal solutions for all instances. Diversity in the construction heuristics was necessary and the four

				Time		Phase				Construction heur.			
Class	runs	opt.	Dev.	avg	max.	P0	P1	P2	P3	H1	H2	H3	H4
u_120	100	100	0	0.00	0.01	30	40	0	30	70	0	0	0
u_250	100	100	0	0.15	3.19	0	55	10	35	100	0	0	0
u_500	100	100	0	0.00	0.01	0	20	75	5	100	0	0	0
u_1000	100	100	0	0.01	0.03	0	0	90	10	100	0	0	0
t_60	100	100	0	0.33	2.53	0	0	0	100	89	9	2	0
t_120	100	100	0	1.14	6.88	0	0	0	100	88	9	3	0
t_249	100	100	0	0.29	2.91	0	0	0	100	100	0	0	0
t_501	100	100	0	1.24	19.26	0	0	0	100	99	1	0	0
set_1	3600	3596	1	0.20	23.89	2735	340	230	291	833	22	4	2
set_2	2400	2400	0	0.01	1.89	1180	300	760	160	1215	0	5	0
set_3	50	50	0	4.71	50.61	0	0	40	10	50	0	0	0
gau_1	85	60	1	0.60	2.40	10	0	50	0	50	0	0	0
was_1	500	500	0	0.02	0.14	0	0	415	85	500	0	0	0
was_2	500	500	0	0.02	3.58	0	0	480	20	500	0	0	0
Total	7935	7906											

TABLE 11. Run statistics for HI_BP with five different seeds each

algorithms were instrumental to lead to optimal solutions for all instances. It is worth noting that optimal solutions were found in 7906 runs out of a total of 7935 (five runs for each of the 1587 test instances). The 29 exceptions occurred in four runs of class set_1 and in 25 runs of gau_1.

These results illustrate the robustness of the hybrid improvement heuristic. We also notice that the lower bound computed by HI_BP does not coincide with the optimal value for only 47 out of the 1587 instances considered in Table 11 (one instance from u_250, 43 instances from set_1, one instance from set_3, and two instances from gau_1). These are also precisely the same instances were the largest computation times were observed.

9. Concluding remarks

Our hybrid improvement procedure for the bin packing problem has several features: the incorporation of lower bounding strategies; the generation of initial solutions by reference to the dual min-max problem; the use of load redistribution based on dominance, differencing, and unbalancing; and an improvement process utilizing tabu search. The move selection strategy used by the tabu search improvement procedure is a major contribution of this work and very likely can be applied to other problems in similar situations.

Encouraging results have been obtained for different sets of benchmark instances, clearly showing the robustness of the algorithm. We improved the best known solution for 11 instances from [29]. Procedure HI_BP also improved the best solutions found by a recent VNS procedure [9] for 41 instances. Also, HI_BP performed much better than MTPCS [27] for a quite large set of difficult instances. We note that the best results previously reported in the literature 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, our algorithm is the only one that has succeeded in finding the best known results for all instances.

An extension of heuristic HI_BP was recently applied to the multiprocessor scheduling problem. Promising computational results will be reported elsewhere.

Acknowledgments: We thank two anonymous referees for several valuable comments that contributed to improve the final presentation of this paper. Work of A. Alvim was supported by doctorate scholarships of CAPES and CNPq and was partially done while she was visiting the Laboratory PRiSM at the University of Versailles, France. The first author is grateful to E. Uchoa for fruitful discussions.

References

[1] A.C.F. Alvim, F. Glover, C.C. Ribeiro, and D.J. Aloise, "Local search for the bin packing problem", *Extended Abstracts of the 3rd Metaheuristics International Conference*, 7–12, Angra dos Reis, 1999.

[2] A.C.F. Alvim, D.J. Aloise, F. Glover, and C.C. Ribeiro, "A hybrid improvement heuristic for the bin packing problem", *Extended Abstracts of the 4th Metaheuristics International Conference*, 63–68, Porto, 2001.

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

[4] M.F. Argüello, T.A. Feo, and O. Goldschmidt, "Randomized methods for the number partitioning problem", *Computers and Operations Research* 23 (1996), 103–111.

[5] E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson, "Approximation algorithms for bin packing: A survey", in *Approximation Algorithms for NP-Hard Problems* (D. Hochbaum, ed.), 46–93, PWS Publishing, 1997.

[6] H. Dyckhoff, "A typology of cutting and packing problems", *European Journal of Operational Research* 44 (1990), 145–159.

[7] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing", *Journal of Heuristics* 2 (1996), 5–30.

[8] S.P. Fekete and J. Schepers, "New classes of fast lower bounds for bin packing problems", *Mathematical Programming* 91 (2001) 1, 11–31.

[9] K. Fleszar and K. Hindi, "New heuristics for one-dimensional bin packing", *Computers and Operations Research* 29 (2002) 7, 821–839.

[10] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W.H. Freeman and Company, 1979.

[11] I. Gent, "Heuristic solution of open bin packing problems", *Journal of Heuristics* 3 (1998), 299-304.

[12] F. Glover and M. Laguna, *Tabu search*, Kluwer Academic Publishers, 1997.

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

[14] S.T. Hackman, M. Magazine, and T. Wee, "Fast, effective algorithms for simple assembly line balancing problems", *Operations Research* 37 (1989), 916–924.

[15] P. Hansen and N. Mladenović, "An introduction to variable neighbourhood search", in *Metaheuristics: Advances and Trends in Local Search Procedures for Optimization* (S. Voss, S. Martello, I.H. Osman, and C. Roucairol, eds.), 433–458, Kluwer, 1999.

[16] R. Hübscher and F. Glover, "Applying tabu search with influential diversification to multiprocessor scheduling", *Computers and Operations Research* 21 (1994), 877–884.

[17] N. Karmarkar and R.M. Karp, "The differencing method of set partitioning", Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley, 1982.

[18] N. Karmarkar and R.M. Karp, "An efficient approximation scheme for the one-dimensional bin packing problem", in *Proceedings of the 23rd. Symposium on Foundations of Computer Science*, 312–320, IEEE Computer Society, 1982.

[19] R.M. Karp, "Reducibility among combinatorial problems", in *Complexity of Computer Computations* (R.E. Miller and J.M. Thatcher, eds.), 85–103, Plenum Press, 1972.

[20] S. Martello and P. Toth, *Knapsack problems: Algorithms and computer implementations*, Wiley, 1990.

[21] S. Martello and P. Toth, "Lower bounds and reduction procedures for the bin packing problem", *Discrete Applied Mathematics* 28 (1990), 59–70.

[22] J. E. Schoenfield, "Fast, exact solution of open bin packing problems without linear programming", Working paper, 2002.
[23] A. Scholl, online document at http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/binpp/index.htm, last visited on April 13, 2003.
[24] A. Scholl, R. Klein, and C. Jürgens, "BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem", *Computers and Operations Research* 24 (1997), 627–645.
[25] A. Scholl and S. Voss, "Simple assembly line balancing - Heuristic approaches",*Journal of Heuristics* 2 (1996), 217–244.
[26] P. Schwerin and G. Wäscher, "The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP", *International Transactions in Operational Research* 4 (1997), 377–389.
[27] P. Schwerin and G. Wäscher, "A new lower bound for the bin-packing problem and its integration into MTP", *Pesquisa Operacional* 19 (1999), 111–129.
[28] N.Y. Soma, H.H. Yanasse, and N. Maculan, "A heuristic for the bin packing problem", presented at the IFORS Triennial Conference, Beijing, 1999.
[29] Special Interest Group on Cutting and Packing, online document at http://www.apdio.pt/sicup/Sicuphomepage/research.htm, last visited on April 13, 2003.
[30] J.M. Valério de Carvalho, "Exact solution of bin-packing problems using column generation and branch-and-bound", *Annals of Operations Research* 86 (1999), 629–659.
[31] F. Vanderbeck, "Computational study of a column generation algorithm for bin packing and cutting stock problems", *Mathematical Programming* 86 (1999), 565–594.
[32] G. Wäscher and T. Gau, "Heuristics for the integer one-dimensional cutting stock problem: A computational study", *OR Spektrum* 18 (1996), 131–144.

(A.C. Alvim) Department of Computer Science, Catholic University of Rio de Janeiro, Rua Marquês de São Vicente 225, Rio de Janeiro, 22453-900, Brazil.
E-mail address, A.C. Alvim: alvim@inf.puc-rio.br

(C.C. Ribeiro) Department of Computer Science, Catholic University of Rio de Janeiro, Rua Marquês de São Vicente 225, Rio de Janeiro, 22453-900, Brazil.
E-mail address, C.C. Ribeiro: celso@inf.puc-rio.br

(F. Glover) Leeds School of Business, University of Colorado at Boulder, Boulder, CO 80309-0419, United States.
E-mail address, F. Glover: Fred.Glover@colorado.edu

(D.J. Aloise) Universidade Federal do Rio Grande do Norte, Department of Computer Science and Applied Mathematics, Natal, RN 59078-970, Brazil.
E-mail address, D.J. Aloise: dario@digi.com.br