INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems

Andrea Lodi, Silvano Martello, Daniele Vigo,

INFORMS Journal on Computing
Vol. 11, No. 4, Fall 1999

0899-1499/99/1104-0345 $05.00
© 1999 INFORMS

Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems

ANDREA LODI, SILVANO MARTELLO, AND DANIELE VIGO / *Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, Viale Risorgimento, 2-40136 Bologna, Italy, Emails: alodi@deis.unibo.it; smartello@deis.unibo.it; dvigo@deis.unibo.it*

(Received: October 1998; revised: May 1999; accepted: July 1999)

Two-dimensional bin packing problems consist of allocating, without overlapping, a given set of small rectangles (items) to a minimum number of large identical rectangles (bins), with the edges of the items parallel to those of the bins. According to the specific application, the items may either have a fixed orientation or they can be rotated by 90°. In addition, it may or not be imposed that the items are obtained through a sequence of edge-to-edge cuts parallel to the edges of the bin. In this article, we consider the class of problems arising from all combinations of the above requirements. We introduce a new heuristic algorithm for each problem in the class, and a unified tabu search approach that is adapted to a specific problem by simply changing the heuristic used to explore the neighborhood. The average performance of the single heuristics and of the tabu search are evaluated through extensive computational experiments.

Informally speaking, a *two-dimensional bin packing problem* (2BP) consists of allocating, without overlapping, a given set of "small" rectangles (*items*) to a minimum number of "large" identical rectangles (*bins*). In general, it is additionally required that the items are packed with their edges parallel to those of the bins. This basic problem has many real-world applications: cutting of standardized stock units in wood or glass industries, packing on shelves or truck beds in transportation and warehousing, paging of articles in newspapers, to mention just a few.

According to the specific application, several variants can arise, but in most cases the additional requirements are the following:

1. *Orientation*. The items may either have a fixed orientation or they can be rotated (by 90°).
2. *Guillotine cuts*. It may or not be imposed that the items are obtained through a sequence of edge-to-edge cuts parallel to the edges of the bin.

The guillotine constraint is frequently present in cutting problems, due to technological characteristics of automated cutting machines, whereas it is generally not imposed in packing applications. Rotation is not allowed in newspaper paging or when the items to be cut are decorated or corrugated, whereas orientation is free in the case of plain materials and in most packing contexts.

In this article we propose a simple typology for the class of bin packing problems defined by the four cases arising from the above two requirements. For each case, we present a new effective heuristic algorithm. We then introduce a unified metaheuristic framework for the whole class, which is adapted to a specific problem by simply changing the heuristic used to explore the neighborhood.

Formally, we have n items, and each item j is defined by a *width* w_j and a *height* h_j ($j = 1, \ldots, n$). An unlimited number of identical bins is available, each having width W and height H. We want to pack all items into the minimum number of bins, in such a way that no two items overlap and the edges of the items are parallel to those of the bins. We assume, without loss of generality, that all input data are positive integers. We consider four problems:

2BP|O|G: the items are oriented (*O*), i.e., they cannot be rotated, and guillotine cutting (G) is required;
2BP|R|G: the items may be rotated by 90° (R) and guillotine cutting is required;
2BP|O|F: the items are oriented and cutting is free (F);
2BP|R|F: the items may be rotated by 90° and cutting is free.

Although not all four variants are equally relevant in industrial contexts, a number of specific applications can be found in the bin packing or cutting stock literature. For example, 2BP|O|G has to be solved in the crepe-rubber cutting described by Schneider;[20] 2BP|R|G is related to the cutting stock problems arising in the steel industry, discussed by Vasko, Wolf, and Stott;[21] 2BP|O|F models the problem of placing advertisements and other material in newspapers and yellow pages, studied by Lagus, Karanta, and Ylä-Jääski;[15] several applications of 2BP|R|F are mentioned by Bengtsson.[2] In the following, an asterisk will be used to denote both variants of a specific field. In order to ensure feasibility, we assume, without loss of generality, that:

1. $h_j \leqslant H$ and $w_j \leqslant W$ ($j = 1, \ldots, n$) for 2BP|O|∗;
2. $\min\{w_j, h_j\} \leqslant \min\{W, H\}$ and $\max\{w_j, h_j\} \leqslant \max\{W, H\}$ ($j = 1, \ldots, n$) for 2BP|R|∗.

In the well-known *one-dimensional bin packing problem* (1BP), one is required to partition n given elements having associated values $h_1, \ldots, h_n$ into the minimum number of subsets so that the sum of the values in each subset does not exceed a prefixed capacity H. Given any instance of this problem,

Subject classifications: Combinatorial algorithms.
Other key words: packing, cutting, heuristic, tabu search.

consider the instance of the two-dimensional bin packing problem defined by n items j having height h_j and width $w_j = 1$ ($j = 1, \ldots, n$), with bin height H and bin width $W = 1$. It is then clear that any of the four variants described above will solve the one-dimensional instance. It follows that our problems are strongly NP-hard, since it is known that the one-dimensional bin packing problem is such.

It is worth mentioning that most of the two-dimensional bin packing problems considered in the literature fall into the above cases. Theoretical contributions usually concern 2BP|O|F: Chung, Garey, and Johnson[5] and Frenk and Galambos[13] have proposed upper bounds with asymptotic worst-case performance guarantee, whereas Martello and Vigo[19] have analyzed lower bounds and presented an exact branch-and-bound approach. The remaining literature is mostly devoted to applications and heuristics concerning one of the four cases. For 2BP|O|F, Berkey and Wang[3] have presented extensions of classical one-dimensional bin packing heuristics. Other heuristics have been proposed by Bengtsson[2] and El-Bouri, Popplewell, Balakrishnan, and Alfa[12] for 2BP|R|F, by Lodi, Martello, and Vigo[16] for 2BP|R|G, and by Lodi, Martello and Vigo[17] for 2BP|O|G. The reader is also referred to Dyckhoff and Finke[10] and Dowsland and Dowsland[8] for general surveys on packing problems, and to Dyckhoff, Scheithauer, and Terno[11] for an annotated bibliography.

We finally observe that Dyckhoff[9] has proposed a typology for cutting and packing problems, which is not however adequate in the present context, since all four problems we treat are classified there as 2/V/I/M.

In the next section, we give basic definitions and briefly review relevant algorithms from the literature. In Section 2, we present a new heuristic algorithm for each of the four variants of 2BP. In Section 3, we introduce a general tabu search scheme that can virtually be used for the solution of any 2BP. The average performance of both heuristic and metaheuristic approaches is experimentally evaluated in Section 4.

1. Basic Definitions and Algorithms

Throughout this article, we denote an algorithm as A_{XY}, where A is the algorithm's name, whereas $X \in \{O, R\}$ and $Y \in \{G, F\}$ indicate the specific problem solved by A. It is worth mentioning that an algorithm for one of the four problems we consider may produce feasible solutions for other problems too. For example, an algorithm A_{OG} always produces solutions that are feasible for all four problems, whereas an algorithm A_{RF} may only be used for 2BP|R|F. The complete set of compatibilities between solutions and problems is depicted in the graph of Figure 1: an arc (v_i, v_j) implies that a solution feasible for the variant associated with vertex v_i is also feasible for the variant associated with vertex v_j.

Several heuristic algorithms from the literature obtain a feasible solution by packing the items in rows forming levels, called *shelves* (see Coffman, Garey, Johnson, and Tarjan[6]). The first shelf of a bin coincides with its bottom; new shelves within the same bin are created along the horizontal line that coincides with the top of the tallest item packed on the highest shelf.

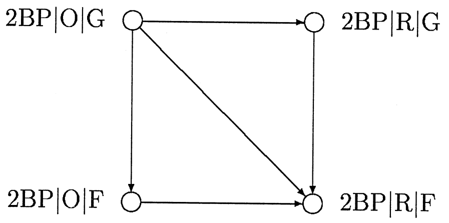

Figure 1. Compatibilities between solutions and problems.

In many cases (see Chung, Garey, and Johnson[5]), the solution is obtained in two phases, the first of which consists in solving an associated problem, known as the *two-dimensional strip packing problem*. In this case, one is given a single open-ended bin of width W and infinite height (*strip*), and the objective is to pack all the items such that the height to which the strip is filled is minimized. In the second phase, the packed strip is subdivided into finite bins of height H. The strip packing is usually obtained through a shelf algorithm, so it is clear that the second phase consists in solving a one-dimensional bin packing problem in which each shelf is viewed as an element having value $\tilde{h}_j$ ($j = 1, \ldots$) equal to the height of the tallest item it contains.

The first approach of this type has been proposed and analyzed (from the worst-case behavior point of view) by Chung, Garey, and Johnson.[5] Effective heuristics have then been derived by Berkey and Wang[3] for 2BP|O|G. Two of their algorithms, FBS_{OG} and FFF_{OG}, will be frequently referred to throughout the article, and are briefly described hereafter. Both initially sort the items by nonincreasing height.

Algorithm *Finite Best Strip* (FBS_{OG}) starts by packing the strip according to a *best-fit decreasing* policy: if the current item does not fit into any existing shelf, then a new shelf is initialized for it; otherwise the current item is packed onto the shelf that minimizes the residual horizontal space. In the second phase, the resulting shelves are packed into finite bins using the best-fit decreasing one-dimensional bin packing heuristic: the next (highest) shelf is packed into the bin that minimizes the residual vertical capacity, or into a new one, if no bin can accommodate it. The algorithm can be implemented so as to have time complexity $O(n \log n)$. As an example, consider the instance shown in Figure 2: the strip packing produced in the first phase is depicted in Figure 2*A*, and the corresponding finite bin solution is given in Figure 2*B*.

Algorithm *Finite First Fit* (FFF_{OG}) directly packs the items into finite bins (skipping the strip packing phase), according to a *first-fit decreasing* policy: the current item is packed into the first bin that can accommodate it, or on the bottom of a new one, if no such bin exists; in the former case the item is packed onto the first (lowest) existing shelf that can accommodate it, or by initializing a new one if no such shelf exists. The algorithm has time complexity $O(n^2)$. For the instance in Figure 2, algorithm FFF_{OG} produces the same solution as FBS_{OG}.

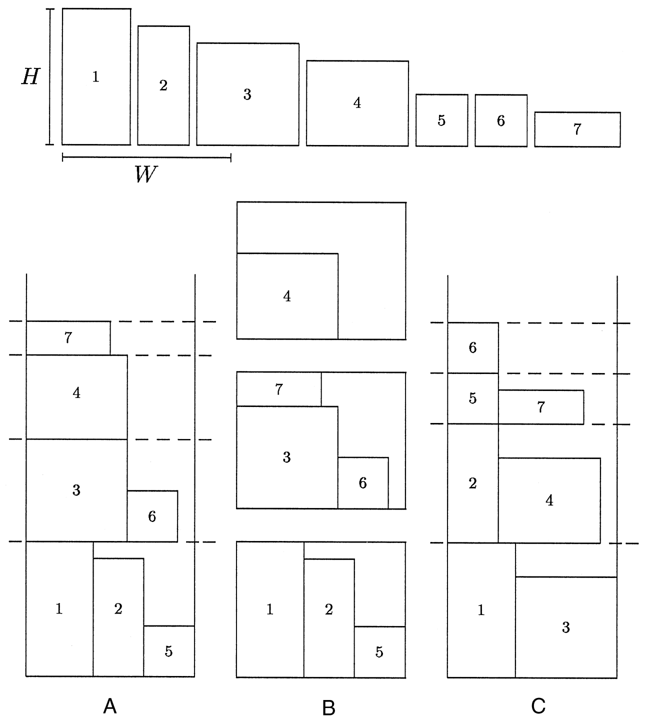

Figure 2. A two-dimensional bin packing instance with $n = 7$, and (*A*) strip packing produced by FBS_{OG}; (*B*) finite bin solution found by FBS_{OG} and FFF_{OG}; (*C*) strip packing produced by KP_{OG} (Section 3.1).

It is easily seen that both algorithms above produce guillotine packings. A sequence of horizontal cuts separates the shelves. For each shelf, a vertical cut produces the left-most item packed onto it, and a series of pairs (vertical cut, horizontal cut) produces the remaining items.

Lodi, Martello, and Vigo[16] have considered adaptations of FBS_{OG} and FFF_{OG} to the case where rotation is allowed. We will say that an item is in *horizontal* (resp. *vertical*) orientation if its longest (resp. shortest) edge is parallel to the bottom of the bin or the strip. Both algorithms, denoted by FBS_{RG} and FFF_{RG} in the following, start by sorting the items according to nonincreasing value of their shortest edge, and by horizontally orienting them. Their iterative phase differs from that of FBS_{OG} and FFF_{OG} in two aspects: 1) when the current item initializes a new shelf, it is always packed horizontally, so as to minimize the height of the shelf; 2) when an existing shelf is considered for possible packing of the current item, if both orientations are feasible, then the vertical one is always considered, so as to favor future packings onto the same shelf. A different evolution of the two-phase shelf algorithms is the following approach proposed by Lodi, Martello, and Vigo[16] for 2BP|R|G.

Algorithm *Floor-Ceiling* (FC_{RG}) initially sorts the items by nonincreasing value of their shortest edge, and horizontally orients them. The main peculiarity is in the way the strip shelves are packed in the first phase. The algorithms described so far place the items, from left to right, with their bottom edge on the bottom line of the shelf (*floor*). Algorithm FC_{RG} may, in addition, place items, from right to left, with their top edge on the shelf *ceiling* (i.e., the horizontal line defined by the top edge of the tallest item packed in the shelf). Whenever an item is packed: 1) the rotated placing is also evaluated; 2) the guillotine constraint is checked and the item placing is possibly modified accordingly. In the second phase, the shelves are packed into finite bins using an exact algorithm for the one-dimensional bin packing problem.

Note that the floor-ceiling approach allows for immediate adaptations to the three remaining problems:

FC_{RF}: drop step 2;

FC_{OG}: initially sort the items according to nonincreasing height, and drop step 1;

FC_{OF}: initially sort the items by nonincreasing height, and drop both steps 1 and 2.

The computational experiments of Section 4.1 prove that the floor-ceiling algorithms outperform, on average, the other approaches from the literature. Better specific algorithms are introduced in the next sections.

We finally observe that the floor-ceiling algorithms above, as well as some of the new algorithms presented in the following sections, require the solution of NP-hard problems, hence their time complexity is non-polynomial. In practice, however, we always halted the execution of the codes for the NP-hard problems after a prefixed (small) number of iterations. The computational experiments showed that, in almost all cases, the optimal solution was obtained before the limit was reached.

2. New Heuristic Algorithms

In this section, we introduce new heuristic algorithms for each of the four variants of the two-dimensional bin packing problem. The algorithms for 2BP|∗|G are directly derived from the shelf approaches by introducing a new way for packing the items on the shelves. The algorithms for 2BP|∗|F are instead based on completely different ideas.

2.1 Oriented Items, Guillotine Cutting (2BP|O|G)

In Section 1, we have seen algorithms FFF_{OG} and FBS_{OG} by Berkey and Wang[3] and the floor-ceiling algorithm FC_{OG}. In the present section, we introduce a new two-phase algorithm in which the shelf packings are determined by solving a series of knapsack problems. In the 0-1 *knapsack problem* (KP01), one is given n elements, each having an associated profit p_j and cost c_j ($j = 1, \ldots, n$), and a capacity q. The problem is to select a subset of elements whose total cost does not exceed q, and whose total profit is a maximum.

The proposed algorithm, denoted by KP_{OG}, at each iteration of the strip packing phase, initializes a new shelf with the tallest unpacked item, say j^*. The shelf packing is then completed by solving an instance of KP01 having an element for each unpacked item j, with profit $p_j = w_j h_j$ and cost $c_j = w_j$, and capacity $q = W - w_{j^*}$. The algorithm can be outlined as follows.

```
algorithm KP_OG:
  sort the items according to nonincreasing $h_j$ values;
  comment: Phase 1;
  repeat
```

```
        open a new shelf in the strip by packing the first
        unpacked item;
        solve the KP01 instance associated with the shelf;
        pack the selected items onto the shelf
    until all items are packed;
    let $\tilde{h}_1, \ldots, \tilde{h}_s$ be the heights of the resulting shelves;
    comment: Phase 2;
    determine a finite bin solution by solving the 1BP
    instance having $s$ elements, with associated values
    $\tilde{h}_1, \ldots, \tilde{h}_s$, and capacity $H$
end.
```

Consider again the example in Figure 2: the strip packing produced by KP_{OG} is given in Figure 2C; the resulting finite bin solution clearly packs the first two shelves in two bins, and the two remaining shelves in a third bin.

It is easily seen that the solutions produced by KP_{OG} always fulfill the guillotine constraint. As previously mentioned, an efficient implementation was obtained by halting the non-polynomial routines for the KP01 and 1BP instances after a prefixed (small) number of iterations.

2.2 Non-Oriented Items, Guillotine Cutting (2BP|R|G)

For this problem, we have already seen in Section 1 the adaptations FFF_{RG} and FBS_{RG} of the Berkey and Wang[3] algorithms, and the floor-ceiling algorithm FC_{RG} by Lodi, Martello, and Vigo.[16] We next describe a new effective algorithm, obtained by extending the knapsack-based approach introduced in the previous section.

The following variations to KP_{OG} are introduced in order to exploit the possibility of rotating the items. The items are initially sorted according to nonincreasing value of their shortest edge, and horizontally oriented: this orientation is always used for the shelf initializations (as in algorithm FC_{RG}).

For each shelf, say of height h^*, the instance of KP01 includes all unpacked items, either in vertical orientation, if no item size exceeds h^*, or in horizontal orientation, otherwise. Note that the profit associated with each item is independent of the orientation, while the cost is reduced whenever the vertical orientation is chosen. Therefore, this strategy ensures that the optimal solution value of the resulting KP01 instance, i.e., the total area packed in the strip, is maximum over the feasible item orientations.

Once a feasible finite bin solution has been obtained from the resulting shelves, as in KP_{OG}, an alternative solution is determined as follows. An instance of 2BP|O|G is built, which has a pseudo-item for each of the previous shelves, with sizes given by the height of the shelf and by the horizontal space actually used in it; for this pseudo-item, the vertical orientation is chosen whenever possible, i.e., if its longest edge does not exceed the finite bin height. Algorithm KP_{OG} is then executed for the resulting instance, and the best of the two final solutions is selected. The overall algorithm follows.

```
algorithm $KP_{RG}$:
    sort the items by nonincreasing $\min\{w_j, h_j\}$ values, and
    horizontally orient them;
    comment: Phase 1;
    repeat
        open a new shelf in the strip by horizontally packing
        the first unpacked item;
        appropriately orient each unpacked item;
        solve the resulting KP01 instance, and pack the
        selected items
    until all items are packed;
    let $\tilde{w}_i$ and $\tilde{h}_i$ $(i = 1, \ldots, s)$ be the sizes of the $s$ resulting
    shelves;
    comment: Phase 2;
    determine a finite bin solution by solving the associated
    1BP instance;
    let $z_1$ be the solution value obtained for the 2BP|R|G
    instance;
    comment: Phase 3;
    define $s$ pseudo-items having sizes $\tilde{w}_i, \tilde{h}_i$ $(i = 1, \ldots, s)$;
    vertically orient each pseudo-item $i$ such that $\max\{\tilde{w}_i, \tilde{h}_i\}$
    $\leqslant H$;
    execute algorithm $KP_{OG}$ for the 2BP|O|G instance
    induced by the pseudo-items;
    let $z_2$ be the resulting solution value;
    set $z := \min\{z_1, z_2\}$
end.
```

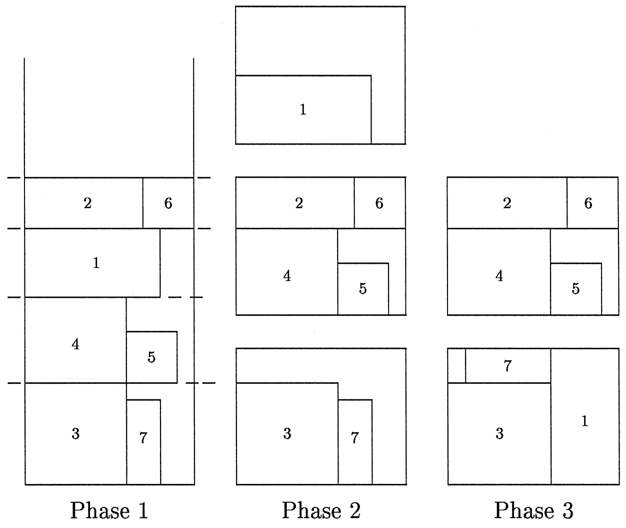

Figure 3. Solutions produced by Algorithm KP_{RG}.

Once again consider the example in Figure 2, but interpret it as an instance of 2BP|R|G. The items are sorted by KP_{RG} as (3, 4, 1, 2, 5, 6, 7), with items 1 and 2 rotated by 90°. Figure 3 shows the solution found by each phase of KP_{RG}. Note that the solution at the end of Phase 2 still requires three bins, while the rotation of pseudo-items in Phase 3 produces an optimal two-bin solution.

As to the time complexity of KP_{RG}, the same considerations made for KP_{OG} obviously apply.

2.3 Oriented Items, Free Cutting (2BP|O|F)

To our knowledge, no specific heuristic for this case has been presented in the literature. In Section 1, we introduced the extension, FC_{OF}, of the floor-ceiling approach. In this

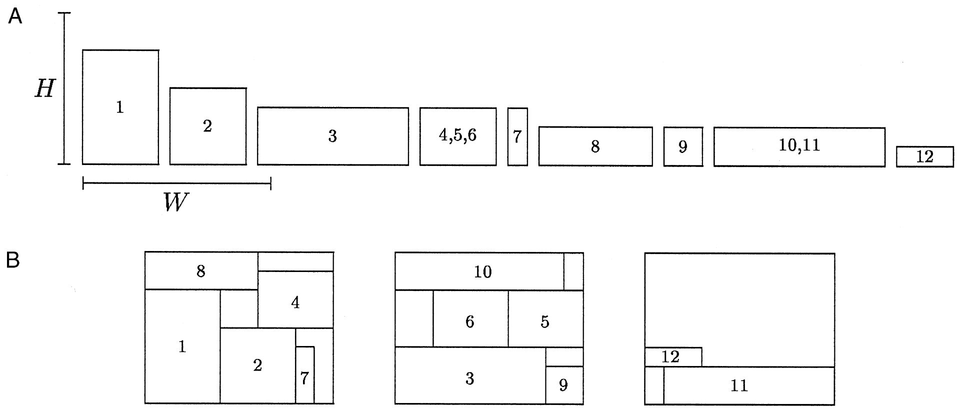

Figure 4. *A*, Two-dimensional bin packing instance with $n = 12$; *B*, solution found by Algorithm AD_{OF}.

section, we propose a new effective heuristic, which no longer uses shelves, but exploits the feasibility of non-guillotine patterns by packing the items into the bins in alternate directions, i.e., from left to right, then from right to left in the lowest possible position, and so on.

More precisely, the *Alternate Directions* (AD_{OF}) algorithm starts by sorting the items according to nonincreasing heights, and by computing a lower bound L on the optimal solution value. (A trivial bound is of course the *continuous* lower bound $L_0 = \lceil \Sigma_{j=1}^{n} w_j h_j/(WH) \rceil$; better bounds can be found in Martello and Vigo.[19]) Then, L bins are initialized by packing on their bottoms a subset of the items, following a best-fit decreasing policy. The remaining items are packed into *bands* according to the current *direction* associated with the bin. Namely, if the direction is "from left to right" (resp. "from right to left"): 1) the first item of the band is packed with its left (resp. right) edge touching the left (resp. right) edge of the bin, in the lowest possible position; 2) each subsequent item is packed with its left (resp. right) edge touching the right (resp. left) edge of the previous item in the band, in the lowest possible position. Observe that the items packed (from left to right) by the initialization step constitute the first band of each bin. The direction associated with all the bins after this step is then "from right to left." In the iterative part of the algorithm, for each bin, we scan all the unpacked items, possibly packing them in the current direction, and changing direction when no further item can be packed in the current band. As soon as no item can be packed in either direction, we move to the next bin, or we initialize a new empty bin. The algorithm can be formally stated as follows.

```
algorithm AD_OF:
  sort the items according to nonincreasing $h_j$ values;
  comment: Phase 1;
  compute a lower bound $L$ on the optimal solution value,
  and open $L$ empty bins;
  for $j := 1$ to $n$ do
    if item $j$ can be packed on the bottom of some bin
    then
      pack $j$, left justified, on the bottom of the bin whose
      residual horizontal space is a minimum;
  comment: Phase 2;
  $i := 0$;
  repeat
    $i := i + 1$, right_to_left := true, n_fail := 0;
    repeat
      let $j$ be the first unpacked item which can be
      packed in bin $i$ according to the current value of
      right_to_left, if any;
      if $j$ = nil then
        n_fail := n_fail + 1,
        right_to_left := not right_to_left
      else
        pack $j$ into bin $i$ according to right_to_left,
        n_fail := 0;
    until n_fail = 2
  until all items are packed
end.
```

As an example, consider the instance of 12 items shown in Figure 4*A*. The bin sizes are $W = 10$ and $H = 8$, and the item sizes are: $w_1 = 4$, $h_1 = 6$; $w_2 = h_2 = 4$; $w_3 = 8$, $h_3 = 3$; $w_4 = w_5 = w_6 = 4$, $h_4 = h_5 = h_6 = 3$; $w_7 = 1$, $h_7 = 3$; $w_8 = 6$, $h_8 = 2$; $w_9 = 2$, $h_9 = 2$; $w_{10} = w_{11} = 9$, $h_{10} = h_{11} = 2$; $w_{12} = 3$; $h_{12} = 1$. Lower bound L_0 gives value 2. Figure 4*B* shows the solution found by AD_{OF}. In Phase 1, items 1, 2, 3, 7, and 9 are packed. In Phase 2, a band is added to the first bin (items {4, 8}), and two to the second one (items {5, 6} and {10}). The third bin is then opened, and packs the remaining items, in two bands. Note that the pattern obtained in the first bin is not guillotine cuttable.

Algorithm AD_{OF} can be implemented so as to run in $O(n^3)$ time. Indeed, the heaviest of the lower bounds described by Martello and Vigo[19] requires $O(n^3)$ time. The main repeat-until loop considers $O(n)$ bins, and, for each of them, $O(n)$ items are examined in the inner loop. For each item, its possible packing can be evaluated in $O(n)$ time, since: 1) the horizontal coordinate at which the item can be placed is unique (see above); 2) the lowest possible vertical coordinate can be determined in $O(n)$ time by checking the interaction of the item with at most $O(n)$ items currently packed in the bin.

2.4 Non-Oriented Items, Free Cutting (2BP|R|F)

We have already seen in Section 1 the adaptation, FC_{RF}, of the floor-ceiling approach. In addition, algorithm AD_{OF} of the previous section can easily be adapted to exploit the possibility of rotating the items. The best computational results were however obtained by the following completely different approach.

The algorithm, called *Touching Perimeter* (TP_{RF}), starts by sorting the items according to nonincreasing area (breaking ties by nonincreasing $\min\{w_j, h_j\}$ values), and by horizontally orienting them. A lower bound L on the optimal solution value is then computed, and L empty bins are initialized. (The continuous lower bound L_0 defined in the previous section is obviously valid for 2BP|R|F as well; better bounds are proposed by Dell'Amico, Martello, and Vigo.[7]) The algorithm packs one item at a time, either in an existing bin, or by initializing a new one. The first item packed in a bin is always placed in the bottom-left corner. Each subsequent item is packed in a so-called *normal position* (see Christofides and Whitlock[4]), i.e., with its bottom edge touching either the bottom of the bin or the top edge of another item, and with its left edge touching either the left edge of the bin or the right edge of another item.

The choice of the bin and of the packing position is done by evaluating a *score*, defined as the percentage of the item perimeter that touches the bin and other items already packed. This strategy favors patterns where the packed items do not "trap" small areas, which may be hard to use for further placements. For each candidate packing position, the score is evaluated twice for the two item orientations (if both are feasible), and the highest value is selected. Score ties are broken by choosing the bin having the maximum packed area. The overall algorithm is as follows.

```
algorithm $TP_{RF}$:
  sort the items by nonincreasing $w_j h_j$ values, and
  horizonally orient them;
  comment: Phase 1;
  compute a lower bound $L$ on the optimal solution value,
  and open $L$ empty bins;
  comment: Phase 2;
  for $j := 1$ to $n$ do
    $score := 0$;
    for each normal packing position in an open bin do
      let $score_1$ and $score_2$ be the scores associated with
      the two orientations;
      $score := \max\{score, score_1, score_2\}$
    end for;
    if $score > 0$ then
      pack item $j$ in the bin, position and orientation
      corresponding to $score$
    else
      open a new bin, and horizontally pack item $j$ into it
  end for
end.
```

Consider again the example in Figure 4, but interpret it as an instance of 2BP|R|F. The items are sorted by TP_{RF} as (1, 3, 10, 11, 2, 4, 5, 6, 8, 9, 7, 12), with items 1 and 7 rotated by 90°.

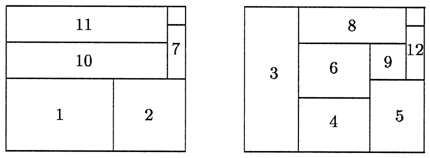

Figure 5. Solution found by Algorithm TP_{RF}.

Figure 5 shows the solution found by TP_{RF}. Note that the pattern in the second bin is not guillotine cuttable.

The iterative part of the algorithm requires $O(n^3)$ time since, for each item, the number of normal positions is $O(n)$ and, for each position, the touching perimeter is computed in $O(n)$ time.

3. A Unified Tabu Search Framework

A metaheuristic approach is an attractive way for guiding the operations of a subordinate heuristic in order to obtain high-quality solutions to a difficult combinatorial optimization problem. Among the different metaheuristic techniques, tabu search recently proved to be particularly effective for two-dimensional bin packing problems (see, e.g., Lodi, Martello, and Vigo[16, 17]). The reader is referred to Aarts and Lenstra[1] and to Glover and Laguna[14] for an introduction to metaheuristic and tabu search algorithms. In this section, we introduce a general tabu search scheme that can virtually be used for the solution of any two-dimensional finite bin packing problem by simply changing the subordinate inner heuristic.

The main feature of our approach is the use of a unified parametric neighborhood, whose size and structure are dynamically varied during the search. The scheme and the neighborhood are independent of the specific problem to be solved. The peculiarities of the problem are taken into account only in the choice of a specific deterministic algorithm to be used, within the neighborhood search, for the evaluation of the moves. In the following, we will denote by A such an algorithm, and by $A(S)$ the solution value it produces when applied to the (sub)instance of 2BP induced by item set S. In Section 4.2, the effectiveness of the proposed scheme is computationally evaluated, on the four packing problems we are considering, using for A the four heuristics introduced in the previous section, and algorithms FBS_{OG} and FBS_{RG} described in Section 1.

Given a current solution, the neighborhood is searched through *moves* that consist in modifying the solution by changing the packing of a subset of items S, in an attempt to empty a specific *target bin*. To this end, subset S always includes one item, j, from the target bin and the current contents of k other bins. The new packing for S is obtained by executing algorithm A on S. In this way, parameter k defines the size and the structure of the current neighborhood. Its value is automatically increased or decreased during the search, and the algorithm maintains k distinct tabu lists. The target bin is selected as the one that is more likely to be emptied by the moves. We adopted the same policy used in the specialized tabu search algorithms presented by

Lodi, Martello, and Vigo,[16, 17] i.e., the target bin t is determined as the one minimizing, over all current bins i, the *filling function*

$$\varphi(S_i) = \alpha \frac{\sum_{j\in S_i} w_j h_j}{WH} - \frac{|S_i|}{n} \tag{1}$$

where S_i denotes the set of items currently packed into bin i, and α is a pre-specified positive weight (equal to 20 in our computational experiments). The resulting choice favors the selection of target bins packing a small area, breaking ties by bins packing a relatively large number of items.

The overall algorithm, 2BP_TABU, is formally stated below. An initial incumbent solution is obtained by executing algorithm A on the complete instance, while the initial tabu search solution consists of packing one item per bin. At each iteration, a target bin is selected, and a sequence of moves, each performed within a procedure SEARCH, tries to empty it. Procedure SEARCH also updates the value of parameter k and, in special situations, may impose performing *diversification actions* (to be discussed later). The execution is halted as soon as a proven optimal solution is found, or a time limit is reached.

```
algorithm 2BP_TABU:
  $z^* := A(\{1, \ldots, n\})$ (comment: incumbent solution value);
  compute a lower bound $L$ on the optimal solution value;
  if $z^* = L$ then stop;
  initialize all tabu lists to empty;
  pack each item into a separate bin;
  $z := n$ (comment: tabu search solution value);
  while time limit is not reached do
    determine the target bin $t$;
    $diversify$ := false, $k := 1$;
    while $diversify$ = false and $z^* > L$ do
      $k_{in} := k$;
      call SEARCH($t,k,diversify,z$);
      $z^* := \min\{z^*, z\}$;
      if $k \leqslant k_{in}$ then determine the new target bin $t$
    end while;
    if $z^* = L$ then stop
    else perform a diversification action
  end while
end.
```

Given the target bin t, procedure SEARCH explores the neighborhood defined by the current value of parameter k. For each item j in bin t we evaluate the candidate moves by executing algorithm A on the subinstances induced by all possible sets S defined by j and by all k-tuples of other bins. In two special situations the move is immediately performed, and the control returns to the main algorithm: 1) when a move decreases the current number of used bins; 2) when a non-tabu move removes j from t by packing the subinstance in exactly k bins. In addition, in these cases, the neighborhood is changed by decreasing the current value of parameter k by one unit.

When neither 1) nor 2) apply, a *penalty* is associated with the move. The penalty is infinity if the move is tabu, or if algorithm A used at least two extra bins (i.e., $A(S) > k + 1$), or if $k = 1$. Otherwise, the penalty is computed as follows. We determine, among the $k + 1$ bins produced by A, the one, say $\bar{t}$, that minimizes the filling function, and execute algorithm A on the subinstance induced by the items in bin $\bar{t}$ plus the residual items in the target bin. If a single bin solution is obtained, the penalty of the overall move is set to the minimum among the filling function values computed for the $k + 1$ resulting bins; otherwise, the penalty is set to infinity.

When a neighborhood has been entirely searched without detecting case 1 or 2 above, the move having the minimum finite penalty (if any) is performed and the control returns to 2BP_TABU. If, instead, the minimum penalty is infinity, i.e., no acceptable move has been found, the neighborhood is changed by increasing the current value of parameter k by one unit, or, if k already reached a maximum prefixed value $k_{\max}$, by imposing a diversification action. The overall procedure can be outlined as follows.

```
procedure SEARCH($t,k,diversify,z$):
  $penalty^* := +\infty$;
  for each $j \in S_t$ do
    for each $k$-tuple $K$ of bins not including $t$ do
      $S := \{j\} \cup (\cup_{i\in K} S_i)$;
      $penalty := +\infty$;
      case
        $A(S) < k$:
          execute the move and update the current
          solution value $z$;
          $k := \max\{1, k - 1\}$;
          return;
        $A(S) = k$:
          if the move is not tabu or $S_t \equiv \{j\}$ then
            execute the move and update the current
            solution value $z$;
            if $S_t \equiv \{j\}$ then $k := \max\{1, k - 1\}$;
            return
          end if;
        $A(S) = k + 1$ and $k > 1$:
          let $I$ be the set of $k + 1$ bins used by $A$;
          $\bar{t} := \arg\min_{i\in I}\{\varphi(S_i)\}$, $T := (S_t\backslash\{j\}) \cup S_{\bar{t}}$;
          if $A(T) = 1$ and the move is not tabu then
            $penalty := \min\{\varphi(T), \min_{i\in I\backslash\{\bar{t}\}}\{\varphi(S_i)\}\}$
      end case;
      $penalty^* := \min\{penalty^*, penalty\}$;
    end for;
  end for;
  if $penalty^* \neq +\infty$ then execute the move corresponding
  to $penalty^*$
  else if $k = k_{\max}$ then $diversify$: = true else $k := k + 1$
return.
```

The diversification actions and the tabu lists remain to be described. The algorithm performs two kinds of diversification, controlled by a counter d, initially set to one. Whenever a diversification is imposed, d is increased by one and the target bin is determined as the one having the d-th smallest value of the filling function. If however $d > z$ or $d = d_{\max}$ (where $d_{\max}$ is a prefixed limit), a stronger diversification is performed, namely: 1) the $\lfloor z/2 \rfloor$ bins with smaller value of the filling function are removed from the tabu search solution; 2) a new solution is obtained by packing alone in a separate bin each item currently packed into a removed bin; 3) all tabu lists are reset to empty, and the value of d is reset to one.

As previously mentioned, there are a tabu list and a tabu tenure τ_k ($k = 1, \ldots, k_{\max}$) for each neighborhood. For $k > 1$, each list stores the *penalty** values corresponding to the last τ_k moves performed, in the corresponding neighborhood, on exit from the two for-each loops of SEARCH. For $k = 1$ instead, since the first and third case of SEARCH can never occur, the moves are only performed when the second case occurs, i.e., without computing a penalty. Hence, the tabu list stores the values of the filling function, $\varphi(S)$, corresponding to the last τ_1 sets S for which a move has been performed. The attributes stored in the tabu lists are real values, considerably varying with the solutions, see Aarts and Lenstra,[1] so this choice is likely to prevent short-term cycling through moves.

4. Computational Experiments

All the algorithms presented in Section 2 and the tabu search framework were coded in FORTRAN 77 and run on a Silicon Graphics INDY R10000sc 195 MHz on test instances from the literature. The solutions of the 1BP and KP01 instances, needed by some of the deterministic algorithms, were obtained, respectively, through FORTRAN codes MTP (with a limit of 300 backtrackings) and MT1 (with a limit of 500 backtrackings) included in the diskette accompanying the book by Martello and Toth.[18]

We considered ten classes of randomly generated problems. The first six classes have been proposed by Berkey and Wang:[3]

Class I: w_j and h_j uniformly random in [1,10], $W = H = 10$;
Class II: w_j and h_j uniformly random in [1,10], $W = H = 30$;
Class III: w_j and h_j uniformly random in [1,35], $W = H = 40$;
Class IV: w_j and h_j uniformly random in [1,35], $W = H = 100$;
Class V: w_j and h_j uniformly random in [1,100], $W = H = 100$;
Class VI: w_j and h_j uniformly random in [1,100], $W = H = 300$.

In each of the above classes, all the item sizes are generated in the same interval. Martello and Vigo[19] have proposed the following classes, where a more realistic situation is considered. The items are classified into four types:

Type 1: w_j uniformly random in [⅔W,W], h_j uniformly random in [1, ½H];
Type 2: w_j uniformly random in [1, ½W], h_j uniformly random in [⅔H, H];
Type 3: w_j uniformly random in [½W, W], h_j uniformly random in [½H, H];
Type 4: w_j uniformly random in [1, ½W], h_j uniformly random in [1, ½H].

The bin sizes are $W = H = 100$ for all classes, while the items are as follows:

Class VII: type 1 with probability 70%, type 2, 3, 4 with probability 10% each;
Class VIII: type 2 with probability 70%, type 1, 3, 4 with probability 10% each;
Class IX: type 3 with probability 70%, type 1, 2, 4 with probability 10% each;
Class X: type 4 with probability 70%, type 1, 2, 3 with probability 10% each.

For each class, we considered five values of n: 20, 40, 60, 80, 100. For each class and value of n, ten instances were generated. In the following two sections we present the computational results obtained for the deterministic algorithms and for the tabu search approach.

It is worth mentioning that test instances where the items sizes are drawn from uniform distributions are usually quite easy to solve for the one-dimensional bin packing problem (see, e.g., Martello and Toth,[18] Ch. 8). This is however not true for the two-dimensional case. For example, an effective branch-and-bound algorithm for 2BP|O|F (see Martello and Vigo[19]) was not able to solve about one-third of the 500 instances described above. Even small size instances are far from being trivial: fifteen percent of the instances with $n = 40$ were not solved to optimality.

4.1 Results for Deterministic Algorithms

The results are presented in Tables I and II. The first two columns give the class and the value of n. The next two pairs of columns refer to the algorithms by Berkey and Wang[3] and give the results for 2BP|O|∗ and 2BP|R|∗, respectively: the algorithms are the original ones in Berkey and Wang[3] (FFF_{OG} and FBS_{OG}) and the variants allowing item rotation described in Section 1 (FFF_{RG} and FBS_{RG}). The following four pairs of columns refer to the four problems considered: for each problem we give the results obtained by the corresponding variant of the floor-ceiling approach described in Section 1 and by the specific heuristic in Section 2.

For each algorithm, the entries report the average ratio (heuristic solution value)/(lower bound), computed over the ten generated instances. For 2BP|O|∗ we used the lower bound by Martello and Vigo,[19] and for 2BP|R|∗ the bound by Dell'Amico, Martello, and Vigo.[7] For each class, the final line gives the average over all values of n. We do not give the CPU times, as they are negligible (never exceeding 0.5 seconds).

By considering, for each class, the average values computed over all values of n, we see that the new algorithms proposed in the present paper outperform, in general, all the other heuristics from the literature. For all cases where this does not happen, the best is the floor-ceiling approach. The most difficult problems, with average errors exceeding 10%, turn out to belong to Classes III, V, and VI for all problem types, to Classes VII and VIII for 2BP|R|∗, and to Class X for 2BP|O|∗.

As to the percentage errors, it is also noteworthy that they are computed with respect to a lower bound value, hence they give a pessimistic estimate of the actual quality of the

Table I. Deterministic Algorithms: (heuristic solution value)/(lower bound); Random Problem Instances Proposed by Berkey and Wang

		Berkey and Wang				Lodi, Martello, and Vigo							
		2BP|O|*		2BP|R|*		2BP|O|G		2BP|R|G		2BP|O|F		2BP|R|F	
		FFF_{OG}	FBS_{OG}	FFF_{RG}	FBS_{RG}	FC_{OG}	KP_{OG}	FC_{RG}	KP_{RG}	FC_{OF}	AD_{OF}	FC_{RF}	TP_{RF}
Class	n	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB
I	20	1.17	1.14	1.09	1.06	1.14	1.13	1.06	1.06	1.12	1.12	1.06	1.05
	40	1.12	1.09	1.10	1.08	1.09	1.10	1.08	1.07	1.08	1.09	1.08	1.06
	60	1.10	1.07	1.12	1.09	1.07	1.07	1.09	1.07	1.07	1.07	1.09	1.05
	80	1.08	1.06	1.10	1.09	1.06	1.06	1.09	1.08	1.06	1.06	1.09	1.06
	100	1.07	1.06	1.08	1.08	1.06	1.05	1.07	1.05	1.06	1.05	1.07	1.03
	Average	1.108	1.084	1.098	1.080	1.084	1.082	1.078	1.066	1.078	1.078	1.078	1.050
II	20	1.10	1.10	1.00	1.00	1.10	1.00	1.00	1.00	1.10	1.00	1.00	1.00
	40	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10	1.10
	60	1.15	1.15	1.10	1.10	1.15	1.15	1.05	1.15	1.10	1.10	1.05	1.00
	80	1.07	1.07	1.07	1.07	1.07	1.07	1.03	1.07	1.07	1.07	1.03	1.07
	100	1.06	1.06	1.03	1.06	1.03	1.03	1.03	1.03	1.03	1.03	1.03	1.00
	Average	1.096	1.096	1.060	1.066	1.090	1.070	1.042	1.070	1.080	1.060	1.042	1.034
III	20	1.20	1.18	1.20	1.20	1.18	1.18	1.18	1.12	1.18	1.20	1.18	1.06
	40	1.18	1.14	1.21	1.16	1.14	1.15	1.16	1.16	1.14	1.15	1.16	1.11
	60	1.14	1.11	1.20	1.18	1.11	1.12	1.19	1.12	1.11	1.13	1.19	1.11
	80	1.13	1.10	1.20	1.15	1.10	1.10	1.15	1.12	1.10	1.10	1.15	1.10
	100	1.12	1.09	1.16	1.14	1.09	1.09	1.13	1.10	1.09	1.09	1.13	1.08
	Average	1.154	1.124	1.194	1.166	1.124	1.128	1.162	1.124	1.124	1.134	1.162	1.092
IV	20	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	40	1.10	1.10	1.10	1.10	1.00	1.10	1.00	1.00	1.00	1.00	1.00	1.00
	60	1.20	1.20	1.10	1.10	1.10	1.20	1.10	1.10	1.10	1.15	1.10	1.10
	80	1.10	1.10	1.10	1.10	1.10	1.13	1.10	1.10	1.10	1.10	1.10	1.07
	100	1.10	1.10	1.07	1.07	1.10	1.10	1.07	1.07	1.10	1.03	1.07	1.03
	Average	1.100	1.100	1.074	1.074	1.060	1.106	1.054	1.054	1.060	1.056	1.054	1.040
V	20	1.14	1.14	1.08	1.08	1.14	1.13	1.08	1.08	1.14	1.14	1.08	1.06
	40	1.11	1.11	1.14	1.12	1.11	1.09	1.10	1.11	1.11	1.11	1.10	1.11
	60	1.11	1.10	1.13	1.11	1.10	1.10	1.11	1.11	1.10	1.10	1.11	1.08
	80	1.12	1.09	1.13	1.10	1.09	1.09	1.11	1.10	1.09	1.09	1.11	1.08
	100	1.12	1.09	1.13	1.10	1.09	1.09	1.10	1.09	1.09	1.09	1.10	1.08
	Average	1.120	1.106	1.122	1.102	1.106	1.100	1.100	1.098	1.106	1.106	1.100	1.082
VI	20	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	40	1.40	1.40	1.40	1.40	1.40	1.50	1.40	1.40	1.40	1.40	1.40	1.40
	60	1.10	1.10	1.05	1.05	1.10	1.10	1.05	1.05	1.10	1.05	1.05	1.05
	80	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	100	1.13	1.10	1.10	1.07	1.10	1.10	1.07	1.10	1.10	1.07	1.07	1.07
	Average	1.126	1.120	1.110	1.104	1.120	1.140	1.104	1.110	1.120	1.104	1.104	1.104

heuristic solution. According to our experience, the gap between lower bound and exact solution can be quite large. Consider for example the instances of 2BP|O|F, for which the branch-and-bound algorithm by Martello and Vigo[19] is available: the average errors computed by using the best available solution (not always optimal) are 1.040, 1.000, 1.062, 1.000, 1.049, 1.000, 1.035, 1.050, 1.005, and 1.034, respectively for Classes I to X, i.e., about one-third, on average, of those reported in the tables.

The use of lower bounds in the algorithms' evaluation needs an additional observation. Since the same instances are solved with different constraints, one could expect a

Table II. Deterministic Algorithms: (heuristic solution value)/(lower bound); Random Problem Instances Proposed by Martello and Vigo

		Berkey and Wang				Lodi, Martello, and Vigo							
		2BP|O|*		2BP|R|*		2BP|O|G		2BP|R|G		2BP|O|F		2BP|R|F	
Class	n	$\frac{FFF_{OG}}{LB}$	$\frac{FBS_{OG}}{LB}$	$\frac{FFF_{RG}}{LB}$	$\frac{FBS_{RG}}{LB}$	$\frac{FC_{OG}}{LB}$	$\frac{KP_{OG}}{LB}$	$\frac{FC_{RG}}{LB}$	$\frac{KP_{RG}}{LB}$	$\frac{FC_{OF}}{LB}$	$\frac{AD_{OF}}{LB}$	$\frac{FC_{RF}}{LB}$	$\frac{TP_{RF}}{LB}$
VII	20	1.10	1.10	1.19	1.19	1.10	1.10	1.19	1.17	1.08	1.10	1.19	1.13
	40	1.11	1.11	1.17	1.17	1.11	1.07	1.17	1.17	1.09	1.10	1.17	1.10
	60	1.08	1.08	1.18	1.18	1.08	1.06	1.18	1.16	1.07	1.07	1.18	1.12
	80	1.07	1.06	1.19	1.17	1.06	1.06	1.17	1.17	1.06	1.06	1.17	1.11
	100	1.04	1.04	1.17	1.17	1.04	1.04	1.17	1.16	1.04	1.04	1.17	1.11
	Average	1.080	1.078	1.180	1.176	1.078	1.066	1.176	1.166	1.068	1.074	1.176	1.114
VIII	20	1.17	1.16	1.18	1.16	1.16	1.12	1.16	1.16	1.16	1.13	1.16	1.16
	40	1.09	1.08	1.19	1.19	1.08	1.07	1.19	1.19	1.07	1.08	1.19	1.16
	60	1.06	1.06	1.18	1.18	1.06	1.06	1.18	1.18	1.06	1.06	1.18	1.11
	80	1.07	1.06	1.17	1.16	1.06	1.05	1.16	1.15	1.06	1.06	1.16	1.11
	100	1.06	1.06	1.17	1.17	1.06	1.04	1.17	1.17	1.06	1.06	1.17	1.12
	Average	1.090	1.084	1.178	1.172	1.084	1.068	1.172	1.170	1.082	1.078	1.172	1.132
IX	20	1.01	1.01	1.00	1.00	1.01	1.01	1.00	1.00	1.01	1.01	1.00	1.01
	40	1.02	1.02	1.01	1.01	1.02	1.02	1.01	1.01	1.02	1.02	1.01	1.02
	60	1.02	1.02	1.01	1.01	1.02	1.01	1.01	1.01	1.02	1.02	1.01	1.01
	80	1.02	1.02	1.01	1.01	1.02	1.02	1.01	1.01	1.02	1.02	1.01	1.01
	100	1.02	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	Average	1.018	1.016	1.008	1.008	1.016	1.014	1.008	1.008	1.016	1.016	1.008	1.012
X	20	1.14	1.14	1.15	1.15	1.14	1.16	1.15	1.12	1.14	1.10	1.15	1.20
	40	1.14	1.09	1.09	1.09	1.09	1.10	1.09	1.09	1.09	1.09	1.09	1.08
	60	1.15	1.12	1.11	1.09	1.10	1.10	1.09	1.08	1.08	1.11	1.09	1.09
	80	1.15	1.13	1.09	1.07	1.12	1.12	1.06	1.06	1.11	1.10	1.06	1.06
	100	1.14	1.10	1.07	1.07	1.10	1.08	1.07	1.05	1.09	1.10	1.07	1.06
	Average	1.144	1.116	1.102	1.094	1.110	1.112	1.092	1.080	1.102	1.100	1.092	1.098

dominance between the algorithms. Indeed, an algorithm for a less-constrained variant produces, on average, better solutions in terms of number of bins. This does not always appear in the average ratios reported in the tables, due to the fact that we use different lower bounds for the four problems. The phenomenon is more evident if one considers the absolute solution values: for example, the total number of bins (over the 500 solved instances) is 7480, 7297, 7487, and 7184 for KP_{OG}, KP_{RG}, AD_{OF}, and TP_{RF}, respectively. The slight anomaly in the behavior of KP_{OG} and AD_{OF} disappears when the algorithms are used within tabu search: the total number of bins becomes then 7433, 7101, 7373, and 7100, respectively.

We finally note that, quite surprisingly, the average ratios for algorithms FC_{RG} and FC_{RF} are identical. Indeed, they always produced solutions using the same number of bins. However, in 164 of 500 instances the packing patterns were actually different, and those produced by FC_{RF} were not guillotine cuttable.

4.2 Results for Tabu Search

The tabu search approach was implemented in a straightforward way from its description in Section 3, i.e., with no additional tailoring or adaptation to the particular problem variant, but the call to the specific inner heuristic algorithm (denoted by A in the pseudo-code). The resulting code is quite compact, consisting of just 500 FORTRAN statements. Tables III and IV present the results it gives, for the four problem variants, when different inner heuristics are invoked.

Good values for the parameters described in Section 3 were experimentally determined as: $\alpha = 20$ (see Eq. 1), $d_{\max} = 50$ (maximum value of the differentiation counter d), $\tau_k = 3$ for all k (tabu tenure). As for $k_{\max}$ (maximum number of distinct tabu lists), we performed experiments with values from 1 to 4, using different time limits: $T_{\lim} = 30, 60, 120, 180$ CPU seconds. The outcome was as follows: 1) for $k_{\max} = 1$ the results were always poor, very seldom improving the starting solution; 2) for $k_{\max} = 2$ the results were quite good

Table III. Tabu Search with Different Inner Heuristics: (tabu search solution value)/(lower bound), CPU Time in Silicon Graphics INDY R10000sc Seconds; Random Problem Instances Proposed by Berkey and Wang

		Berkey and Wang				Lodi, Martello, and Vigo							
		2BP|O|*: heur. FBS_{OG}		2BP|R|*: heur. FBS_{RG}		2BP|O|G: heur. KP_{OG}		2BP|R|G: heur. KP_{RG}		2BP|O|F: heur. AD_{OF}		2BP|R|F: heur. TP_{RF}	
Class	n	$\frac{TS}{LB}$	time	$\frac{TS}{LB}$	time	$\frac{TS}{LB}$	time	$\frac{TS}{LB}$	time	$\frac{TS}{LB}$	time	$\frac{TS}{LB}$	time
I	20	1.09	36.00	1.06	24.00	1.11	42.00	1.03	12.39	1.06	24.00	1.05	18.00
	40	1.08	54.00	1.06	45.90	1.08	54.17	1.05	36.06	1.06	36.11	1.04	30.02
	60	1.05	54.05	1.08	54.24	1.05	54.07	1.05	38.60	1.04	48.93	1.04	34.00
	80	1.04	44.18	1.07	55.44	1.04	42.32	1.07	54.59	1.05	48.17	1.06	48.08
	100	1.04	60.37	1.05	60.24	1.05	60.19	1.04	55.65	1.04	60.81	1.03	47.92
	Average	1.060	49.72	1.064	47.96	1.066	50.55	1.048	39.46	1.050	43.60	1.044	35.60
II	20	1.10	0.01	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01
	40	1.10	0.01	1.10	0.01	1.10	0.01	1.10	0.01	1.10	0.01	1.10	0.01
	60	1.15	0.02	1.10	0.01	1.15	0.06	1.15	0.06	1.10	0.09	1.00	0.01
	80	1.07	12.00	1.07	12.00	1.07	12.00	1.03	6.24	1.07	12.00	1.03	6.10
	100	1.06	12.00	1.03	6.03	1.03	6.00	1.03	6.00	1.03	6.00	1.00	0.01
	Average	1.096	4.80	1.060	3.61	1.070	3.61	1.062	2.46	1.060	3.62	1.026	1.22
III	20	1.18	48.00	1.12	30.00	1.18	48.00	1.09	24.00	1.20	54.00	1.06	18.00
	40	1.12	60.00	1.15	60.00	1.12	60.00	1.11	48.19	1.11	54.02	1.09	42.17
	60	1.08	48.03	1.11	60.02	1.07	49.48	1.10	60.02	1.05	45.67	1.08	54.15
	80	1.07	54.03	1.12	60.03	1.08	57.24	1.07	60.05	1.08	54.31	1.07	60.07
	100	1.09	60.10	1.10	60.11	1.09	60.09	1.08	60.19	1.09	60.10	1.07	60.18
	Average	1.108	54.03	1.120	54.03	1.108	54.96	1.090	50.49	1.106	53.62	1.074	46.91
IV	20	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01
	40	1.10	0.01	1.10	0.01	1.10	0.01	1.00	0.01	1.00	0.01	1.00	0.01
	60	1.20	0.03	1.10	0.01	1.20	0.09	1.10	0.04	1.15	0.14	1.10	0.09
	80	1.10	18.00	1.10	18.00	1.10	18.06	1.03	6.28	1.10	18.00	1.07	12.00
	100	1.10	18.00	1.07	12.00	1.10	18.00	1.03	6.10	1.03	6.00	1.03	6.00
	Average	1.100	7.21	1.074	6.00	1.100	7.23	1.032	2.48	1.056	4.83	1.040	3.62
V	20	1.13	42.00	1.06	18.00	1.13	42.00	1.04	12.01	1.11	36.02	1.04	12.01
	40	1.09	48.00	1.10	42.00	1.09	48.00	1.07	42.01	1.04	27.07	1.07	42.00
	60	1.06	55.59	1.09	60.01	1.07	58.26	1.07	48.72	1.06	56.77	1.06	45.23
	80	1.06	60.05	1.10	60.05	1.08	60.07	1.08	54.60	1.06	56.18	1.07	54.14
	100	1.08	60.14	1.10	60.12	1.09	60.17	1.07	60.26	1.08	60.34	1.07	60.12
	Average	1.084	53.16	1.090	48.04	1.092	53.70	1.066	43.52	1.070	47.28	1.062	42.70
VI	20	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01
	40	1.40	0.01	1.40	0.01	1.50	0.02	1.40	0.02	1.40	0.03	1.40	0.03
	60	1.10	0.01	1.05	0.01	1.10	0.05	1.05	0.02	1.05	0.04	1.05	0.05
	80	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01
	100	1.10	18.00	1.07	12.00	1.10	18.00	1.07	12.11	1.07	12.00	1.07	12.00
	Average	1.120	3.60	1.104	2.40	1.140	3.61	1.104	2.43	1.104	2.41	1.104	2.42

with $T_{\lim} = 30$, slightly improving with higher time limits; 3) for $k_{\max} = 3$ and $T_{\lim} = 30$ the results were slightly worse than for $k_{\max} = 2$, considerably improved with $T_{\lim} = 60$, and very marginally improved with higher time limits; 4) for $k_{\max} = 4$ we had practically no improvement with respect to $k_{\max} = 3$. Hence, we adopted the best tradeoff, i.e., $k_{\max} = 3$ and $T_{\lim} = 60$. As an example, we report in Figure 6 the results obtained on the fifty instances of 2BP|R|F for Class VII, which turned out to be quite sensitive to the parameter variations.

The computational results are presented in Tables III and IV. As in the previous section, the first two columns give the

Table IV. Tabu Search with Different Inner Heuristics: (tabu search solution value)/(lower bound), CPU Time in Silicon Graphics INDY R10000sc Seconds; Random Problem Instances Proposed by Martello and Vigo

		Berkey and Wang				Lodi, Martello, and Vigo							
		2BP|O|∗: heur. FBS_{OG}		2BP|R|∗: heur. FBS_{RG}		2BP|O|G: heur. KP_{OG}		2BP|R|G: heur. KP_{RG}		2BP|O|F: heur. AD_{OF}		2BP|R|F: heur. TP_{RF}	
Class	n	$\frac{TS}{LB}$	time	$\frac{TS}{LB}$	time	$\frac{TS}{LB}$	time	$\frac{TS}{LB}$	time	$\frac{TS}{LB}$	time	$\frac{TS}{LB}$	time
VII	20	1.08	24.01	1.15	42.06	1.08	24.00	1.11	30.00	1.04	12.02	1.11	30.00
	40	1.07	42.06	1.17	60.00	1.07	42.00	1.07	44.97	1.06	37.01	1.08	48.06
	60	1.05	36.68	1.16	60.01	1.05	36.49	1.06	54.30	1.05	36.44	1.06	59.45
	80	1.05	60.08	1.17	60.06	1.05	60.09	1.08	60.20	1.04	54.52	1.10	60.12
	100	1.03	48.40	1.16	60.18	1.04	49.31	1.07	60.71	1.03	47.43	1.08	60.36
	Average	1.056	42.25	1.162	56.46	1.058	42.38	1.078	50.04	1.044	37.48	1.086	51.60
VIII	20	1.12	36.00	1.16	42.00	1.12	36.00	1.10	30.04	1.06	18.04	1.10	30.01
	40	1.04	25.34	1.19	60.00	1.04	24.68	1.08	51.22	1.03	18.72	1.10	54.22
	60	1.03	30.90	1.17	60.04	1.03	30.61	1.07	48.41	1.02	20.99	1.07	56.17
	80	1.03	44.02	1.16	60.13	1.03	45.87	1.08	60.23	1.02	37.95	1.08	60.11
	100	1.04	54.36	1.17	60.18	1.04	54.22	1.08	60.48	1.04	52.66	1.09	60.14
	Average	1.052	38.12	1.170	56.47	1.052	38.28	1.082	50.08	1.034	29.67	1.088	52.13
IX	20	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.01	1.00	0.06
	40	1.01	24.01	1.01	18.00	1.01	24.02	1.01	18.02	1.01	24.05	1.01	18.85
	60	1.01	24.09	1.01	18.14	1.01	24.15	1.01	18.10	1.01	24.26	1.01	18.03
	80	1.02	54.44	1.01	30.24	1.01	48.61	1.01	30.52	1.01	54.31	1.01	30.51
	100	1.01	31.97	1.01	30.68	1.01	30.60	1.01	30.93	1.01	34.11	1.01	36.86
	Average	1.010	26.90	1.008	19.41	1.008	25.48	1.008	19.51	1.008	27.35	1.008	20.86
X	20	1.14	24.00	1.12	6.00	1.14	24.00	1.12	6.00	1.10	12.00	1.12	6.01
	40	1.09	36.00	1.07	32.32	1.09	36.03	1.08	30.87	1.06	25.18	1.06	24.01
	60	1.08	48.03	1.07	40.44	1.08	48.05	1.07	36.73	1.07	42.13	1.06	30.44
	80	1.10	60.02	1.06	48.03	1.10	60.02	1.06	45.68	1.06	47.30	1.05	39.04
	100	1.08	60.10	1.06	54.57	1.07	60.13	1.05	42.28	1.08	60.10	1.05	43.38
	Average	1.098	45.63	1.076	36.27	1.096	45.65	1.076	32.31	1.074	37.34	1.068	28.58

class and the value of n. The next pairs of columns refer to algorithms FBS_{OG} (Berkey and Wang[3]), FBS_{RG} (Section 1), and to the four heuristics in Section 2, when used within the tabu search. For each algorithm, the entries give the average ratio (tabu search solution value)/(lower bound), computed over the ten generated instances, and the average CPU time. In this case too, for each class, the final line gives the average over all values of n.

The results are satisfactory. By considering, for each class, the average values computed over all values of n, we can observe that the tabu search generally improves the initial deterministic solution produced by the inner heuristic, with almost all exceptions occurring for Classes II, IV, and VI. This is not surprising since, for the instances in these classes, depending on the value of n, it is either very easy or very hard to prove the optimality of a presumably optimal heuristic solution. Consider for example Class II, for which the average area of an item is 30.25, hence, on average, 29.75 items are packed in each bin: for $n = 20$, 40, 80, and 100 it is then easy to find an optimal solution using one, two, three, and four bins, respectively, while for $n = 60$ it is very difficult to find a feasible solution using two bins (if any).

In this case, too, the percentage errors could be much smaller if computed with respect to the optimal solution values. For the instances of 2BP|O|F, the average errors computed by using the best available solution (not always optimal) are 1.009, 1.000, 1.035, 1.000, 1.014, 1.000, 1.004, 1.004, 1.000, and 1.008, respectively for Classes I to X, i.e., about one-eighth, on average, of those in the tables.

The computational results presented in this section show that tabu search is effective in all cases, regardless of the inner deterministic algorithm used in the search. This proves, for 2BP, the quality of a unified tabu search approach, capable to further improve the performance of effective deterministic heuristics.

Acknowledgments

We acknowledge the support given to this project by the Ministero dell'Università e della Ricerca Scientifica e Tecnologica

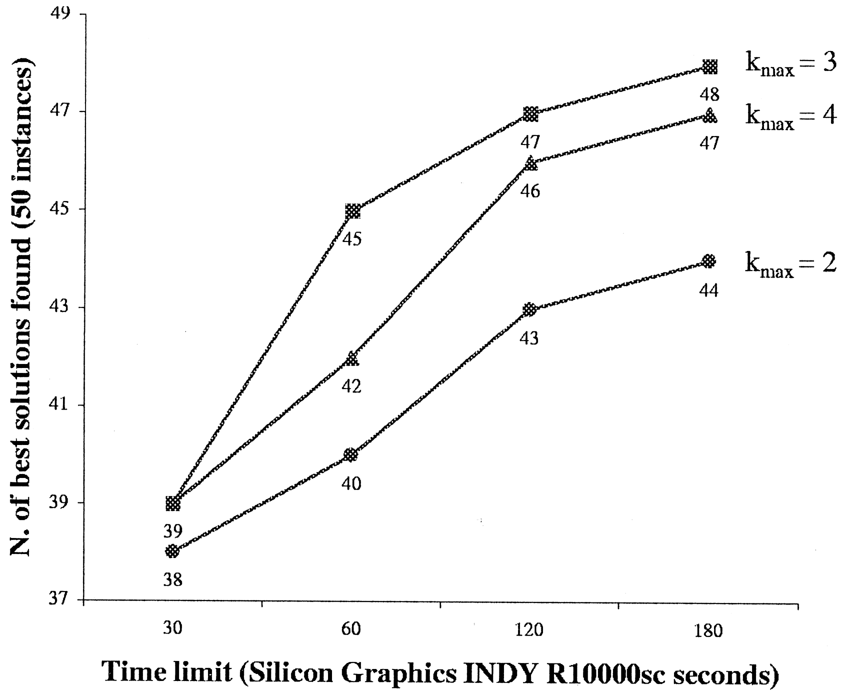

Figure 6. Results of the tabu search for different values of k_{max} and T_{lim} on the instances of Class VII for 2BP|R|F (heuristic TP_{RF}).

(MURST) and the Consiglio Nazionale delle Ricerche (CNR), Italy. We thank two anonymous referees for helpful comments.

References

1. E. AARTS and J.K. LENSTRA (eds.), 1997. *Local Search in Combinatorial Optimization*, John Wiley & Sons, Chichester.
2. B.E. BENGTSSON, 1982. Packing Rectangular Pieces—A Heuristic Approach, *The Computer Journal 25*, 353–357.
3. J.O. BERKEY and P.Y. WANG, 1987. Two Dimensional Finite Bin Packing Algorithms, *Journal of the Operational Research Society 38*, 423–429.
4. N. CHRISTOFIDES and C. WHITLOCK, 1977. An Algorithm for Two-Dimensional Cutting Problems, *Operations Research 25*, 30–44.
5. F.K.R. CHUNG, M.R. GAREY, and D.S. JOHNSON, 1982. On Packing Two-Dimensional Bins, *SIAM Journal of Algebraic and Discrete Methods 3*, 66–76.
6. E.G. COFFMAN, JR., M.R. GAREY, D.S. JOHNSON, and R.E. TARJAN, 1980. Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms, *SIAM Journal on Computing 9*, 801–826.
7. M. DELL'AMICO, S. MARTELLO, and D. VIGO, 1999. An Exact Algorithm for Non-Oriented Two-Dimensional Bin Packing Problems, in preparation.
8. K.A. DOWSLAND and W.B. DOWSLAND, 1992. Packing Problems, *European Journal of Operational Research 56*, 2–14.
9. H. DYCKHOFF, 1990. A Typology of Cutting and Packing Problems, *European Journal of Operational Research 44*, 145–159.
10. H. DYCKHOFF and U. FINKE, 1992. *Cutting and Packing in Production and Distribution*, Physica Verlag, Heidelberg.
11. H. DYCKHOFF, G. SCHEITHAUER, and J. TERNO, 1997. Cutting and Packing (C&P), in *Annotated Bibliographies in Combinatorial Optimization*, M. Dell'Amico, F. Maffioli, and S. Martello, (eds.), John Wiley & Sons, Chichester, 393–413.
12. A. EL-BOURI, N. POPPLEWELL, S. BALAKRISHNAN, and A. ALFA, 1994. A Search Based Heuristic for the Two-Dimensional Bin-Packing Problem, *INFOR 32*, 265–274.
13. J.B. FRENK and G.G. GALAMBOS, 1987. Hybrid Next-Fit Algorithm for the Two-Dimensional Rectangle Bin-Packing Problem, *Computing 39*, 201–217.
14. F. GLOVER and M. LAGUNA, 1997. *Tabu Search*, Kluwer Academic Publishers, Boston.
15. K. LAGUS, I. KARANTA, and J. YLÄ-JÄÄSKI, 1996. Paginating the Generalized Newspaper: A Comparison of Simulated Annealing and a Heuristic Method, in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, Berlin, 549–603.
16. A. LODI, S. MARTELLO, and D. VIGO, 1998. Neighborhood Search Algorithm for the Guillotine Non-Oriented Two-Dimensional Bin Packing Problem, in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I.H. Osman, and C. Roucairol, (eds.), Kluwer Academic Publishers, Boston, 125–139.
17. A. LODI, S. MARTELLO, and D. VIGO, 1999. Approximation Algorithms for the Oriented Two-Dimensional Bin Packing Problem, *European Journal of Operational Research 112*, 158–166.
18. S. MARTELLO and P. TOTH, 1990. *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, Chichester.
19. S. MARTELLO and D. VIGO, 1998. Exact Solution of the Two-Dimensional Finite Bin Packing Problem, *Management Science 44*, 388–399.
20. W. SCHNEIDER, 1988. Trim-Loss Minimization in a Crepe-Rubber Mill; Optimal Solution Versus Heuristic in the 2 (3)-Dimensional Case, *European Journal of Operational Research 34*, 273–281.
21. F.J. VASKO, F.E. WOLF, and K.L. STOTT, 1989. A Practical Solution to a Fuzzy Two-Dimensional Cutting Stock Problem, *Fuzzy Sets and Systems 29*, 259–275.