MULTI-DIMENSIONAL PACKING BY TABU SEARCH

ANDREA LODI

Abstract. The paper deals with multi-dimensional cutting and packing and metaheuristic algorithms. A brief overview of this area is presented, and the general Tabu Search framework of Lodi, Martello and Vigo [LMV99a] is considered in detail. Computational results on multi-dimensional bin packing problems are reported, and possible methods for extending the framework to other packing problems are finally discussed.

Keywords: cutting and packing, Metaheuristic, Tabu search.

1. Introduction

In several industrial applications one is required to allocate a set of rectangular items to larger rectangular standardized stock units by minimizing the waste. In wood or glass industries, rectangular components have to be cut from large sheets of material. In warehousing contexts, goods have to be placed on shelves. In newspapers paging, articles and advertisements have to be arranged in pages. In these applications, the standardized stock units are rectangles, and a common objective function is to pack all the requested items into the minimum number of units: the resulting optimization problems are known in the literature as two-dimensional bin packing problems.

In other contexts, such as paper or cloth industries, we have instead a single standardized unit (a roll of material with virtually infinite height), and the objective is to obtain the items by using the minimum roll length: the problems are then referred to as two-dimensional strip packing problems. In other industrial applications a profit is also associated to each rectangular item and the objective is to allocate a subset of the items without overlapping on a single but finite stock unit so as to maximize the overall profit of the cut items. This is the case, for example, of the steel industry (the problem often emerges as subproblem in various productive processes), and these problems are known in the literature as two-dimensional knapsack problems.

Most of the contributions in the literature are devoted to the case where the items to be packed have a fixed orientation with respect to the stock unit(s),
i.e., one is not allowed to rotate them. This case reflects a number of practical contexts, such as the cutting of corrugated or decorated material (wood, glass, cloth industries), or the newspapers paging. However, many other variants, also reflecting real-world applications, have been investigated in the literature such as the classical rotation version (usually by 90°) and/or the variants arising when constraints on the item placements (e.g., the “guillotine cuts”) are imposed.

General surveys on cutting and packing problems can be found in Dyckhoff and Finke [DF92], Dowsland and Dowsland [DD92] and Dyckhoff, Scheithauer and Terno [?].

Let us introduce the problems in a more formal way. We are given a set of \( n \) rectangular items \( j \in J = \{1, \ldots, n\} \), each defined by a width, \( w_j \), and a height, \( h_j \):

- in the Two-Dimensional Bin Packing Problem (2BP), we are further given an unlimited number of identical rectangular bins of width \( W \) and height \( H \), and the objective is to allocate all the items to the minimum number of bins;
- in the Two-Dimensional Strip Packing Problem (2SP), we are further given a bin of width \( W \) and infinite height (hereafter called strip), and the objective is to allocate all the items to the strip by minimizing the height to which the strip is used;
- in the Two-Dimensional Knapsack Problem (2KP), we are further given a unique and finite bin of width \( W \) and height \( H \), a profit \( p_j \) for each item \( j \in J \), and the objective is to allocate a subset of the items so as to maximize the overall profit (sum of the profits of the selected items).

In all cases, the items have to be packed with their \( w \)-edges parallel to the \( W \)-edge of the bins (or strip). We will assume, with no loss of generality, that all input data are positive integers, and, by restricting to the case in which no rotation is allowed, that \( w_j \leq W \) and \( h_j \leq H \ (j = 1, \ldots, n) \).

The above problems are strongly NP-hard, as can be easily seen by transformation from the strongly NP-hard (one-dimensional) Bin Packing Problem (1BP), in which \( n \) items, each having an associated size \( h_j \), have to be partitioned into the minimum number of subsets so that the sum of the sizes in each subset does not exceed a given capacity \( H \).

Obvious extensions of the described problems arise in the Three-Dimensional case, i.e., when the items to be allocated have an additional size, the depth, \( d_j \) \((j \in J)\), and the same holds for the stock unit(s), say \( D \) (where, obviously, also \( d_j \leq D, \forall j \in J \) is satisfied). Hence, we have:

- the Three-Dimensional Bin Packing (3BP);
Multi-Dimensional Packing by Tabu Search

- the Three-Dimensional Strip Packing;
- the Three-Dimensional Knapsack.

Although exact approaches for some of the discussed problems have been proposed in the literature (and surprisingly many in the last five years), heuristic algorithms have always been the most classical choice in the cutting and packing area. Starting from the early eighties, a large amount of greedy-type heuristics have been proposed and theoretically and/or experimentally evaluated, often by following the seminal work of Johnson [J73] in the 1BP context. Indeed, many of these algorithms are based on adaptations and combinations of the classical next-fit, first-fit and best-fit decreasing algorithms (see Johnson [J73]), and are, in general, fast and quite easy to implement.

Moreover, in the last decade metaheuristic algorithms have shown their effectiveness in solving a large variety of Combinatorial Optimization problems, and considerably efforts have been devoted to cutting and packing.

The present paper deals with multi-dimensional cutting and packing and metaheuristic algorithms and is organized as follows. In the next section we overview the metaheuristic approaches for multi-dimensional cutting and packing which, in our view, give a methodological contribution, i.e., which describe general techniques. In Section 3 we discuss in more detail the general Tabu Search framework proposed by Lodi, Martello and Vigo [LMV99a] in the multi-dimensional bin packing context by also reporting computational results. Finally, in Section 4 we show how the framework can be extended to work for many other multi-dimensional cutting and packing problems, and some conclusions are drawn in Section 5. Since some of the names used in the paper for indicating the considered problems are not uniformly accepted, supplementary names used in other papers are reported and discussed in Appendix.

2. Metaheuristics for multi-dimensional packing

The impact of metaheuristic techniques on the practical solution of multi-dimensional cutting and packing problems has been quite impressive. In this section we briefly review papers on the subject by concentrating, due to the wideness of the area and to the large number of application-oriented papers, on those giving a stronger methodological contribution, i.e., suggesting techniques which can be applied to different categories of cutting and packing.

---

1 The reader is referred to Lodi, Martello and Vigo [LMV99b] for a survey.
2 We refer the reader to Aarts and Lenstra [AL97] for a general introduction to this area.
Dowsland [D93] presented one of the first metaheuristic approaches to 2SP. His simulated annealing algorithm explores both feasible solutions and solutions in which some of the items overlap. During the search, the objective function to be minimized is thus the total pairwise overlapping area, and the neighborhood contains all the solutions corresponding to either vertical or horizontal item shiftings. As soon as a new feasible solution improving the current one is found, an upper bound is fixed to its height. Few computational experiments on small-size instances are reported.

Jakobs [J96] proposed a genetic algorithm for 2SP. His approach is based on a representation of a packing pattern by means of a permutation giving the order in which the items are packed, while the packing positions are determined through the Bottom-Left strategy: an item is always packed in the lowest possible position, left justified (see Chazelle [C83]). This representation turns out to be particularly useful in genetic algorithms for an effective implementation of crossover and mutation operators. Few numerical examples are reported.

Parada, Sepúlveda, Solar and Gómes [PSSG98] presented a simulated annealing algorithm for the 2KP problem in which the profit of each item is equal to its area, thus maximizing the overall profit coincides to minimizing the waste. The algorithm is based on a binary tree representation of the current solution (obtained by greedy-type heuristics). Each node is either a pattern to be cut, say $B$, (the root node is the initial stock unit) or a cut item, say $G$, (thus, the node is a leaf of the tree), and a $B$ node which is not anymore “divided” becomes a waste area, i.e., a type $P$ node. The sum of the areas of $P$ nodes is the objective function to be minimized, and a move is a “modification” of the tree by randomly changing either a $B$ or a $P$ node. Several ways of obtaining this modification are proposed, and additional requirements such as rotation of the items are considered. Computational experiments also on large-size instances are reported.

An extension of the approach by Dowsland [D93] to 2BP and 3BP was recently proposed by Færø, Pisinger and Zachariasen [FPZ99]. They use similar neighborhood and search strategy within a guided local search approach (see Voudouris and Tsang [VT99] for details). Given a lower bound and an upper bound on the optimal solution value, if these do not coincide, the algorithm randomly assigns the items packed in the highest numbered bin to the other bins. The new solution is generally not feasible, so the new objective function is the total pairwise overlap, plus a term that penalizes, during the search, “unlikely” infeasible patterns. The neighborhood is explored through item shiftings. Minimizing the new objective function corresponds to finding a feasible solution that involves one less bin. The process is iterated until either the upper bound
becomes equal to the lower bound, or a prefixed time limit is reached. Special-
ized techniques to reduce the time complexity for the exploration of this very
large neighborhood are implemented. Extensive computational experiments
for both 2BP and 3BP are reported (using the same sets of instances discussed
in Section 3.1).

3. A general dBP Tabu Search framework

Lodi, Martello and Vigo [LMV99a, LMV00] developed effective Tabu Search
(TS) algorithms\(^3\) for 2BP and 3BP. We describe here the unified tabu search
framework given in [LMV99a], whose main characteristic is the adoption of
a search scheme and a neighborhood which are independent of the specific
packing problem to be solved.

Given a current solution, the moves modify it by changing the packing of a
subset \(S\) of items, trying to empty a specified target bin. Let \(S_i\) be the set of
items currently packed into bin \(i\): the target bin \(t\) is the one minimizing, over all
bins \(i\), a so-called filling function: a function giving a measure of the easiness
of emptying the bin. Lodi, Martello and Vigo [LMV99a, LMV00] proposed
for 2BP and 3BP the functions:

\[
\varphi(S_i) = \alpha \frac{\sum_{j \in S_i} w_j h_j}{WH} - \frac{|S_i|}{n}
\]

\[
\varphi(S_i) = \alpha \frac{\sum_{j \in S_i} w_j h_j d_j}{WHD} - \frac{|S_i|}{n}
\]

where \(\alpha\) is in both cases a (possibly different) pre-specified positive weight.
The idea is to favor target bins packing a small area and a relatively large
number of items.

Once the target bin has been selected, subset \(S\) is defined so as to include one
item, \(j\), from the target bin and the current contents of \(k\) other bins. The new
packing for \(S\) is obtained by executing an appropriate greedy-type heuristic
on \(S\). The value of parameter \(k\), which defines the size and the structure of the
current neighborhood, is automatically updated during the search.

If the move packs the items of \(S\) into \(k\) (or less) bins, i.e., item \(j\) has been
removed from the target bin, the move is performed and the value of \(k\) updated
as \(k = \min\{1, k-1\}\) (the target bin is possibly re-computed). Then, a new item
is selected, a new set \(S\) is defined accordingly, and a new move is performed.
Otherwise, \(S\) is changed by selecting a different set of \(k\) bins, or a different
item \(j\) from the target bin (if all possible configurations of \(k\) bins have been
attempted for the current \(j\)).

\(^3\)We refer the reader to Glover and Laguna [GL97] for a specific treatment of this technique.
If the algorithm gets stuck, i.e., the target bin is not emptied, the neighborhood is enlarged by increasing the value of $k$ up to a prefixed upper limit. There are a tabu list and a tabu tenure $\tau_k$ for each value of $k$. A tabu list stores, for each forbidden move, the sum of the filling function values of the $k + 1$ involved bins, and the last $\tau_k$ moves for each value of $k$ are simultaneously stored.

This strategy of reducing and enlarging the size of the neighborhood can be seen as a sort of Variable Neighborhood Search (see Mladenovic and Hansen [MH97]). Small values of $k$ obviously correspond to “small” neighborhoods which are preferred because are fast to explore, but instead of accepting moves involving more than the current number of bins, the neighborhood is enlarged (or changed) so as to have more chances of improving the current solution. The algorithm automatically plays “diversifications” and “intensifications” by varying the size of the neighborhood.

The outer loop of the described scheme is depicted in Figure 3.1. The inner loop, i.e., procedure “SEARCH” in Figure 3.1, explores the neighborhood of the current solution $z$ which is identified by the target bin $t$, and the size $k$. Formally, for each item $j \in S_t$ and for each $k$-tuple of bins different from $t$, a sub-instance involving $j$ plus the items in the $k$-tuple of bins is solved through algorithm $A$ (tentative move). Then, either (i) the first move whose overall number of bins is less than $z$ is selected, or (ii) the best move among those whose overall number of bins is equal to $z$ (i.e., the one involving a bin with the smallest filling function value) is chosen. If neither case (i) nor (ii) arise, the size $k$ of the neighborhood is increased (up to a prefixed limit $k_{\text{max}}$), while in case (i), the value of $k$ is reduced (if it was not already equal to 1)$^4$.

Two more aggressive diversification actions (diversify in Figure 3.1) are also performed. The first one (“soft” diversification) simply consists in selecting as target bin the one having the second smallest filling function value (see (1) and (2) for 2BP and 3BP, respectively). The second one (“hard” diversification) consists in re-packing into separate bins the items currently packed in the $[z/2]$ bins ($z$ being the number of bins in the current solution) with smallest $\varphi(S_t)$ value. In both cases, all tabu lists are reset to empty, and the search is restarted with $k = 1$.

An initial incumbent solution is obtained by executing heuristic $A$ on the complete instance, while the initial tabu search solution consists of packing one item per bin. The execution is halted as soon as a proven optimal solution is found, or a time limit is reached.

The framework, originally proposed for 2BP [LMV99a], has been adapted to 3BP [L00, LMV00], the major changes being the filling function (2) and

$^4$A pseudo-code of “SEARCH” is given in [LMV99a].
algorithm dBP.TABU:
begin
\( z^* := A(\{1, \ldots, n\}) \) \((\text{comment: incumbent solution value})\);
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;
\( z := n \) \((\text{comment: tabu search solution value})\);
while time limit is not reached do
\begin{align*}
determine the target bin \( t \);
diversify := \text{false}; k := 1; \\
\text{while diversify = false and } z^* > L \text{ do} \\
k_{in} := k; \\
call SEARCH(t,k,diversify,z); \\
\min z^* := \min\{z^*,z\}; \\
\text{if } k \leq k_{in} \text{ then determine the new target bin } t \\
end while; \\
\text{if } z^* = L \text{ then stop else perform a diversification action} \\
end while
\end{align*}
end.
}

Figure 3.1: The Tabu Search framework for multi-dimensional bin packing.

the greedy-type heuristic \( A^5 \). Indeed, the bin packing problem to be solved just concerns these two aspects plus some implementation details and running-time parameters. The framework can then be used for virtually any variant of multi-dimensional bin packing problem, by simply changing \( \varphi \) and \( A \), and in [LMV99a] it has been used to solve the variants of 2BP involving the additional constraint of using guillotine cuts (see Section 4) and/or allowing rotation by \( 90^\circ \).

3.1. 2BP and 3BP computation

Since the framework described in the previous section is based on the use of a greedy-type heuristic, a good way of testing the impact of the TS is to measure the improvement with respect to this heuristic. In this section, we report computational experiments on both 2BP and 3BP in which the performance of greedy-type heuristics and Tabu Search are compared.

\(^5\)More details on the adaptation are given in Lodi [L00].
Two-Dimensional Bin Packing: The benchmark consists of 500 random instances\(^6\), with \(n \in \{20, 40, 60, 80, 100\}\). Ten different classes of instances were used. The first four classes were proposed by Martello and Vigo [MV98], and are based on the generation of items of four different types:

- **type 1**: \(w_j \) uniformly random in \([\frac{2}{3} W, W]\), \(h_j \) uniformly random in \([1, \frac{1}{2} H]\);
- **type 2**: \(w_j \) uniformly random in \([1, \frac{1}{2} W]\), \(h_j \) uniformly random in \([\frac{2}{3} H, H]\);
- **type 3**: \(w_j \) uniformly random in \([\frac{1}{2} W, W]\), \(h_j \) uniformly random in \([\frac{1}{2} H, H]\);
- **type 4**: \(w_j \) uniformly random in \([1, \frac{1}{2} W]\), \(h_j \) uniformly random in \([1, \frac{1}{2} H]\).

Class \(k\) (\(k = 1, \ldots, 4\)) is then obtained by generating an item of type \(k\) with probability 70%, and of the remaining types with probability 10% each. The bin size is always \(W = H = 100\).

The next six classes have been proposed by Berkey and Wang [BW87]:

- **Class 5**: \(W = H = 10\), \(w_j \) and \(h_j \) uniformly random in \([1, 10]\);
- **Class 6**: \(W = H = 30\), \(w_j \) and \(h_j \) uniformly random in \([1, 10]\);
- **Class 7**: \(W = H = 40\), \(w_j \) and \(h_j \) uniformly random in \([1, 35]\);
- **Class 8**: \(W = H = 100\), \(w_j \) and \(h_j \) uniformly random in \([1, 35]\);
- **Class 9**: \(W = H = 100\), \(w_j \) and \(h_j \) uniformly random in \([1, 100]\);
- **Class 10**: \(W = H = 300\), \(w_j \) and \(h_j \) uniformly random in \([1, 100]\).

In Table 1 we report the results of the Tabu Search algorithm using three different greedy-type heuristics. Specifically, we tested the classical Hybrid Best-Fit algorithm (HBP) (see Berkey and Wang [BW87]), and two “new” heuristics proposed by Lodi, Martello and Vigo [LMV99a] and called Knapsack Packing (KP) and Alternate Directions (AD). For each pair \((n, \text{Class})\), we report in Table 1 the average number of bins (over the ten instances) needed by the greedy-type heuristics and by the version of the TS using those heuristics. The TS is executed for 60 CPU seconds on a Silicon Graphics INDY R10000sc (195 MHz).

---

\(^6\)Available at http://www.or.deis.unibo.it/research_pages/ORinstances/2BP.html.
Table 1: Impact of the TS framework on the solution of 2BP.

<table>
<thead>
<tr>
<th>n</th>
<th>Class</th>
<th>HBF</th>
<th>TS(HBF)</th>
<th>KP</th>
<th>TS(KP)</th>
<th>AD</th>
<th>TS(AD)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>5.8</td>
<td>5.7</td>
<td>5.8</td>
<td>5.7</td>
<td>5.8</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6.3</td>
<td>6.1</td>
<td>6.1</td>
<td>6.1</td>
<td>6.2</td>
<td>5.8</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>14.4</td>
<td>14.3</td>
<td>14.4</td>
<td>14.3</td>
<td>14.4</td>
<td>14.3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4.5</td>
<td>4.5</td>
<td>4.6</td>
<td>4.5</td>
<td>4.3</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7.6</td>
<td>7.3</td>
<td>7.5</td>
<td>7.4</td>
<td>7.5</td>
<td>7.1</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1.1</td>
<td>1.1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>5.4</td>
<td>5.5</td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>6.8</td>
<td>6.7</td>
<td>6.7</td>
<td>6.7</td>
<td>6.8</td>
<td>6.6</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>11.9</td>
<td>11.5</td>
<td>11.5</td>
<td>11.5</td>
<td>11.8</td>
<td>11.4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>12.0</td>
<td>11.9</td>
<td>11.5</td>
<td>11.5</td>
<td>12.0</td>
<td>11.4</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>27.9</td>
<td>27.8</td>
<td>28.0</td>
<td>27.8</td>
<td>27.9</td>
<td>27.8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>7.7</td>
<td>7.7</td>
<td>7.8</td>
<td>7.7</td>
<td>7.7</td>
<td>7.5</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>13.9</td>
<td>13.8</td>
<td>14.0</td>
<td>13.8</td>
<td>13.9</td>
<td>13.5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>10.0</td>
<td>9.8</td>
<td>10.1</td>
<td>9.8</td>
<td>10.1</td>
<td>9.7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>12.5</td>
<td>12.3</td>
<td>12.4</td>
<td>12.3</td>
<td>12.5</td>
<td>11.9</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1.9</td>
<td>1.9</td>
<td>2.0</td>
<td>2.0</td>
<td>1.9</td>
<td>1.9</td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>16.7</td>
<td>16.3</td>
<td>16.4</td>
<td>16.3</td>
<td>16.6</td>
<td>16.2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16.9</td>
<td>16.4</td>
<td>16.8</td>
<td>16.4</td>
<td>16.9</td>
<td>16.2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>44.0</td>
<td>43.8</td>
<td>43.9</td>
<td>43.7</td>
<td>44.0</td>
<td>43.8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>10.8</td>
<td>10.5</td>
<td>10.7</td>
<td>10.5</td>
<td>10.7</td>
<td>10.4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>20.7</td>
<td>20.3</td>
<td>20.6</td>
<td>20.3</td>
<td>20.7</td>
<td>20.1</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2.8</td>
<td>2.8</td>
<td>2.8</td>
<td>2.8</td>
<td>2.7</td>
<td>2.7</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>14.7</td>
<td>14.3</td>
<td>14.8</td>
<td>14.2</td>
<td>15.0</td>
<td>14.0</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
<td>2.6</td>
<td>2.6</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>18.9</td>
<td>18.3</td>
<td>18.9</td>
<td>18.5</td>
<td>18.9</td>
<td>18.2</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.3</td>
<td>2.2</td>
<td>2.2</td>
</tr>
<tr>
<td>80</td>
<td>1</td>
<td>23.6</td>
<td>23.4</td>
<td>23.6</td>
<td>23.5</td>
<td>23.7</td>
<td>23.2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>23.6</td>
<td>22.8</td>
<td>23.2</td>
<td>22.9</td>
<td>23.6</td>
<td>22.6</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>57.8</td>
<td>57.8</td>
<td>57.8</td>
<td>57.8</td>
<td>57.8</td>
<td>57.7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>13.9</td>
<td>13.5</td>
<td>13.7</td>
<td>13.5</td>
<td>13.5</td>
<td>13.0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>28.5</td>
<td>27.9</td>
<td>28.5</td>
<td>27.9</td>
<td>28.6</td>
<td>28.2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>20.2</td>
<td>19.7</td>
<td>20.3</td>
<td>19.8</td>
<td>20.3</td>
<td>19.8</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>3.3</td>
<td>3.3</td>
<td>3.4</td>
<td>3.3</td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>25.7</td>
<td>25.1</td>
<td>25.6</td>
<td>25.4</td>
<td>25.8</td>
<td>25.1</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Table 1 shows that the TS allows a good improvement in the quality of the solution obtained by the greedy-type heuristics: the TS algorithm solves the 500 instances with 113, 67 and 139 bins less than heuristics HBF, KP and AD, respectively.

Three-Dimensional Bin Packing: The benchmark consists of eight classes of instances proposed by Martello, Pisinger and Vigo [MPV99] by somehow extending the above described random generation.

For Classes 1–5, the bin size is \( W = H = D = 100 \) and the following five types of items are considered:

- **type 1**: \( w_j \) uniformly random in \([1, \frac{1}{2} W] \), \( h_j \) in \([\frac{2}{3} H, H] \), \( d_j \) in \([\frac{1}{2} D, D] \);
- **type 2**: \( w_j \) uniformly random in \([\frac{2}{3} W, W] \), \( h_j \) in \([1, \frac{1}{2} H] \), \( d_j \) in \([\frac{1}{2} D, D] \);
- **type 3**: \( w_j \) uniformly random in \([\frac{2}{3} W, W] \), \( h_j \) in \([\frac{2}{3} H, H] \), \( d_j \) in \([1, \frac{1}{2} D] \);
- **type 4**: \( w_j \) uniformly random in \([\frac{1}{2} W, W] \), \( h_j \) in \([\frac{1}{2} H, H] \), \( d_j \) in \([\frac{1}{2} D, D] \);
- **type 5**: \( w_j \) uniformly random in \([1, \frac{1}{2} W] \), \( h_j \) in \([1, \frac{1}{2} H] \), \( d_j \) in \([1, \frac{1}{2} D] \);

for Class \( k \) (\( k = 1, \ldots, 5 \)), each item is of type \( k \) with probability 60\%, and of the other four types with probability 10\% each. Classes 6–8 are as follows:

- **Class 6**: bin size \( W = H = D = 10 \); \( w_j, h_j, d_j \) uniformly random in \([1, 10] \);
- **Class 7**: bin size \( W = H = D = 40 \); \( w_j, h_j, d_j \) uniformly random in \([1, 35] \);
- **Class 8**: bin size \( W = H = D = 100 \); \( w_j, h_j, d_j \) uniformly random in
For each class, 40 instances were solved, ten for each value of \( n \in \{ 50, 100, 150, 200 \} \), yielding a total of 320 instances\(^7\).

In Table 2 we report the results of the Tabu Search algorithm using the greedy-type heuristic called \textit{Height first-Area second (HA)} proposed by Lodi, Martello and Vigo [LMV00]. The entries of Table 2 are the same of Table 1, but the TS is executed for 60 CPU seconds on a Digital Alpha 533 MHz.

\begin{table}[h]
\centering
\begin{tabular}{cccc}
\hline
\( n \) & Class & HA & TS(HA) \\
\hline
50 & 1 & 13.9 & 13.4 \\
   & 2 & 14.2 & 13.8 \\
   & 3 & 14.0 & 13.4 \\
   & 4 & 29.4 & 29.4 \\
   & 5 & 9.5  & 8.4  \\
   & 6 & 10.5 & 9.9  \\
   & 7 & 8.0  & 7.5  \\
   & 8 & 9.9  & 9.3  \\
100 & 1 & 27.6 & 27.1 \\
   & 2 & 27.0 & 25.8 \\
   & 3 & 27.1 & 26.3 \\
   & 4 & 59.0 & 59.0 \\
   & 5 & 15.8 & 15.1 \\
   & 6 & 20.0 & 19.3 \\
   & 7 & 13.3 & 12.6 \\
   & 8 & 19.9 & 19.0 \\
200 & 1 & 52.7 & 51.9 \\
   & 2 & 51.4 & 50.5 \\
   & 3 & 52.1 & 50.6 \\
   & 4 & 119.0 & 118.8 \\
   & 5 & 28.6 & 28.0 \\
   & 6 & 39.1 & 38.2 \\
   & 7 & 25.2 & 24.6 \\
   & 8 & 31.6 & 31.0 \\
\hline
\end{tabular}
\caption{Impact of the TS framework on the solution of 3BP.}
\end{table}

Also Table 2 shows a good behavior of the TS which uses 201 bins less than algorithm HA to pack the 320 3BP instances of the benchmark.

\section{4. Extending the TS framework}

The core of the Tabu Search framework described in the previous sections is the re-combination of sets of items defined by one item in a specific (“target”) bin and the items in a \( k \)-tuple of different bins. Hence, although it is very general, the framework seems to be restricted to bin packing problems. In fact, we

\(^7\)Available at http://www.diku.dk/~pisinger/codes.html. The same instances have been used by Færø, Pisinger and Zachariasen [FPZ99], see Section 2.
will show in the following examples how the framework can be used for multi-dimensional packing problems which do not involve multiple stock units.

**Multi-dimensional Knapsack:**

A classical way of taking into account the application-oriented constraint of cutting (packing) the items through guillotine cuts, i.e., edge-to-edge cuts parallel to the edges of the stock unit, is to cut the items in *levels*, i.e., rows of items forming layers (see Figure 4.2 for examples of level packing/cutting).

![Figure 4.2: Examples of level packing/cutting.](image)

In the two-dimensional context this was already proposed by Gilmore and Gomory [GG65] who introduced specific level cutting problems known as 2- and 3-staged cutting. Both 2- and 3-staged cutting are special cases of the *k*-staged cutting in which each item has to be obtained by at most *k* stages of cuts. Figures 4.2(a) and 4.2(b) are also examples of 2-staged cutting patterns: 4.2(b) represents the so-called *exact case*, while 4.2(a) is the *non-exact case* since a supplementary cut is permitted to separate each item from a trim area.

On the other hand, many of the greedy-type heuristics in the literature work by levels: for example, the Hybrid Best-Fit algorithm used in Section 3.1 packs, in a first phase, the items by levels in a single strip of infinite length, and

---

8For details on *k*-staged cutting see Hifi [H99].
Multi-Dimensional Packing by Tabu Search

allocates, in a second phase, these levels into the bins. It is easy to see that in the 2BP context this second phase involves the (possibly heuristic) solution of a 1BP in which: (i) the capacity is equal to $H$, and (ii) each level obtained in the first phase corresponds to a pseudo-item of the 1BP with size equal to the height of the level, i.e., to the height of the tallest item obtained by it (e.g., item 1 in the first level of Figure 4.2(a)).

Two-dimensional knapsack problems in which the packing is done by levels can be faced by using the following adaptation of the TS framework of Section 3.

1. A greedy-type heuristic, say $A'$, packing the items into levels is used.
2. A feasible solution is obtained by solving (possibly heuristically) a classical Knapsack Problem (KP01) such that: (i) the capacity is equal to $H$, and (ii) each level computed by $A'$ corresponds to a pseudo-item of the KP01 with weight equal to the height of the level and profit computed as the sum of the profit of each item packed in the level (also taking into account the filling of the level itself, i.e., the packed area).
3. A move is performed by re-combining through $A'$ one item packed in a “target” level which does not belong to the current solution (i.e., the level is not packed into the stock unit) and the items in a $k$-tuple of levels of the current solution.
4. The feasibility of the move is again tested by performing the algorithm for KP01 of point 2 on the restricted instance, while its effectiveness is measured as the sum of the profits of the packed items.
5. Worsening moves can be accepted, and the value of $k$ is increased, for example, after a prefixed number of non-improving iterations.

Despite a few implementation details, the spirit of the framework is conserved, so as its generality: by replacing $A'$ with a greedy-type heuristic for three-dimensional level packing, it can be easily extended to solve three-dimensional knapsack problems, and the same holds for the rotation case or the cases arising when additional constraints are imposed on the packing/cutting of the levels.

The discussed adaptation is currently under testing and appropriate descriptions of the “target” function and of the way the neighborhood’s size is updated can be found in Lodi, Martello and Vigo [LMV01].

Multi-dimensional Strip Packing:

It is easy to see that the above adaptation can be easily extended to the case where the strip is packed by levels. Indeed, a greedy-type algorithm as the one of point 1 above directly returns a feasible solution with objective function
value equal to the sum of the heights of the packed levels. A “target” level can be appropriately defined with the aim of improving the current solution by packing its items in other levels.

Also in this case the resulting algorithm follows the original framework and can take into account through the greedy-type heuristic additional constraints. Appropriate descriptions of possible implementations are given in Lodi, Martello and Vigo [LMV01].

A final example of the use of the TS framework for other cutting and packing problems is given in the 2SP context where it is executed (in its original form) to obtain intermediate solutions. This is the case of the approach proposed by Iori, Martello and Monaci [IMM01] for the general 2SP case, i.e., with no guillotine cuts required. More formally, a set of 2BP problems are solved through the TS framework (halted to a prefixed small number of iterations) by defining the bin sizes as $(W, H_i)$, where $H_i$ takes integer values in the range $[\max_{j \in J} h_j, z^* - 1]$, $z^*$ being the incumbent solution value. For each value $H_i$, the bins obtained in this way are packed with appropriate order in the strip, and a general 2SP solution is obtained by re-allocating some of the items by local search moves (shifting them down and left).

5. Conclusion

In this paper we reviewed the use of metaheuristic algorithms for multi-dimensional cutting and packing by considering in particular the general Tabu Search framework proposed by Lodi, Martello and Vigo [LMV99a] in the bin packing context. The main idea of the framework is to isolate the information concerning the specific packing problem to be solved (dimension, additional constraints or requirements), and let a greedy-type heuristic to take care of the structure and construct feasible solutions. The Tabu Search is then used to drive the search through the solution space by re-combining the packed items, and exploring a neighborhood whose size is automatically updated to alternate “intensification” and “diversification”. This results in a very general algorithm for multi-dimensional bin packing which can also be extended to work in similar fashion for problems involving a single stock unit.

Acknowledgments

The author thanks the Consiglio Nazionale delle Ricerche (CNR), Italy, for the support to the project. Thanks are also due to Mhand Hifi for reading a preliminary version of the paper and suggesting several improvements.
Appendix

Unfortunately, terminology and notation on the wide domain of cutting and packing have always been quite unclear and certainly non-unified. The aim of this appendix is to collect a list of names supplementary used in several papers on cutting and packing by also trying to indicate equivalences and differences. The list is however not exhaustive.

In the two-dimensional context:

- 2BP is often referred to as “Two-Dimensional Cutting Stock Problem”, and mainly when multiple copies of each item are present (the same holds for the one-dimensional case). The name “Two-Dimensional Bin Packing” is also used for the more general case in which the size of the bins are different;
- 2SP is often indicated as “Two-Dimensional Layout Problem”. This latter name is used either when the items have “regular”, i.e., rectangular, and “irregular” shapes;
- 2KP is also called “Unconstrained Two-Dimensional Cutting Stock Problem”.

In the three-dimensional context:

- 3BP is sometimes referred to as “Three-Dimensional Cutting Problem”, mainly when guillotine patterns are required. For the general case, the problem is also known as “Container Loading”;
- 3SP is often known as “Pallet Loading”;
- 3KP is sometimes indicated as “General Unconstrained Cutting Stock Problem”, but also “Container Loading”.

References


SIU 2001


Authors addresses:

Andrea Lodi, Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna Viale Risorgimento, 2 - 40136 - Bologna (Italy).

mailto:alodi@deis.unibo.it