TS^2PACK: A Two-Level Tabu Search for the Three-dimensional Bin Packing Problem

Teodor Gabriel Crainic
Département de management et technologie
École des sciences de la gestion, U.Q.A.M.
and CIRRELT, Montreal, Canada
theo@crt.umontreal.ca

and

Guido Perboli and Roberto Tadei
Control and Computer Engineering Department
Politecnico di Torino, Torino, Italy
guido.perboli@polito.it, roberto.tadei@polito.it

Abstract

Three-dimensional orthogonal bin packing is a problem NP-hard in the strong sense where a set of boxes must be orthogonally packed into the minimum number of three-dimensional bins. We present a two-level tabu search for this problem. The first level aims to reduce the number of bins. The second optimizes the packing of the bins. This latter procedure is based on the Interval Graph representation of the packing, proposed by Fekete and Schepers, which reduces the size of the search space. We also introduce a general method to increase the size of the associated neighborhoods, and thus the quality of the search, without increasing the overall complexity of the algorithm. Extensive computational results on benchmark problem instances show the effectiveness of the proposed approach, obtaining better results compared to the existing ones.

Keywords: Three-dimensional packing, Tabu Search, bin packing.

1 Introduction

Given a set of rectangular-shaped items $i \in I$, with sizes w_i, l_i, and h_i, and an unlimited number of containers of fixed sizes W, L, and H, called bins, the *Three-Dimensional orthogonal Bin Packing* problem consists in orthogonally packing the items into the minimum number of bins. We assume items cannot be rotated. According to the typology introduced by Wäscher et al. [29], the problem can be characterized as the Three-Dimensional Single Bin-Size Bin Packing Problem ($3D - SBSBPP$).

Multi-dimensional Bin Packing (BP) problems have been studied mainly in their 2D versions, the focus on problems in higher dimensions, 3D versions in particular, being quite recent ([24], [9] and [25]). It is interesting to notice that most approaches used to address two-dimensional packing problems cannot be extended to their three-dimensional counterparts or, in the best case, that such an extension yields packing that underuses the volumes of the bins ([24], [23], [25], [19], and [21]). Methods specifically designed for 3D problems must therefore be developed.

$3D - SBSBPP$ is NP-hard in the strong sense. Meta-heuristics appear therefore as the methods of choice when large instances must be addressed. Meta-heuristics proposed in the literature use a composite solution representation, which considers simultaneously the assignments of items to the bins and the packing representation describing the positioning of items inside the bins. This considerably reduces the flexibility and performance of the methods. In fact, in these cases can be necessary to rewrite and adapt the overall heuristics every time a packing constraint changes. This paper addresses this issue and proposes a flexible meta-heuristic that outperforms existing methods.

The main contribution of the paper is the introduction of TS^2PACK, a two-level heuristic aimed at solving $3D - SBSBPP$. TS^2PACK separates the search for the optimal number of bins, related to the assignment of the items to the bins (first-level heuristic), from the optimization of the accommodation of items within bins (second-level heuristic). This results into a significantly more flexible procedure than existing heuristics. Thus, the accommodation heuristic can be modified to take into account additional constraints or use a different solution representation without changing the first level heuristic.

One of the main challenges of Multi-Dimensional Packing problems is efficiently verifying the feasibility of the packings, i.e., given the items assigned to a container and their positions within the container according to a set of orthogonal axes, the assignment is feasible if and only if the items of each pair do not overlap. The second level heuristic uses an implicit representation of a packing in a multi-dimensional environment following the graph-theoretical characterization developed by Fekete and Schepers [10, 11]. According to the literature (Section 2), it is the first time that this representation, which reduces the size of the solution space of the accommodation heuristic, is applied within a meta-heuristic context. TS^2PACK also includes the k-chain-moves algorithm, a general method to increase the accuracy of a neighborhood. It dynamically increases the size of a neighborhood without changing its computational complexity, introducing cycles of changes inside the solution.

Extensive computational results on benchmark problem instances show that the TS^2PACK meta-heuristic outperforms other methods for solving $3D - SBSBPP$. The experimental results also show that k-chain-moves and the graph-theoretical characterization of the solution improve significantly the results of the TS^2PACK heuristic both in quality and accuracy.

The paper is organized as follows. We recall the literature on solution methods for $3D - SBSBPP$ in Section 2. Section 3 is dedicated to introducing the TS^2PACK heuristic. Experimental results are discussed in Section 4.

2 Literature Review

According to several papers and surveys (e.g., [25], [19] and [21]), one of the main issues in building a solution of a multi-dimensional packing problem is finding an efficient and accurate representation of placement of the items inside the container. However, in the following, we will focus on the solution methods for $3D - SBSBPP$, being available a recent review of the packing representations in [7].

The first exact method to solve $3D - SBSBPP$ was proposed by Martello, Pisinger and Vigo [24]. It is a two-level Branch & Bound. The first search tree assigns the items to the bins. For each node of the first-level search tree, they use a second Branch & Bound to verify (prove) whether the items assigned to each bin can be actually packed into it. To perform this verification phase, the authors consider that, given a set of items placed into a container, the resulting set of possible packing is dominated by the packing where the items cannot be moved leftward, downwards or backwards. Furthermore, given a partial packing, there is a limited number of points within the residual space of the container where we can accommodate a new item without the new packing being dominated by another. These points, called Corner Points (CPs), are used by the authors to reduce the search space of the second level Branch & Bound. Martello, Pisinger and Vigo test their procedure on 6 sets of instances with up to 200 items.

In the same paper, the authors derive the first lower bounds for $3D - SBSBPP$. Their best bound considers the items with width and height larger than fixed values p and q, respectively, and determines the subsets of items that, for geometric reasons, cannot be placed side by side.

Fekete and Schepers [10, 11] define an implicit representation of the packing by means of Interval Graphs (IGs), the *Packing Class* (PC) representation. In their work, the authors consider the relative position of the boxes in a feasible packing and, from the projection of the items on each orthogonal axis, they define a graph describing the overlappings of the items in the container. More formally, let $G_d(V, E_d)$ be the interval graph associated to the d^{th} axis with a vertex associated to each item i in the container and a non-oriented edge (i, j) between two items i and j if and only if their projections on axis d overlap. The authors prove necessary conditions on the interval graphs to define a feasible packing. This characterization is used by the authors to develop a two-level tree search. Their computational results, mainly limited to $2D$ problems, show that their method outperforms previous methods. Unfortunately, up to now no comparison with the Branch & Bound by Martello, Pisinger, and Vigo [24] has been performed, even if the approach by Fekete and Schepers could be easily extended to $3D - SBSBPP$.

A new class of lower bounds has been introduced by Fekete and Schepers [10]. The authors extend the use of dual feasible functions, first introduced by Johnson [18], to two and three dimensional packing problems, including $3D - SBSBPP$.

The most recent lower bound, proposed by Boschetti [5], introduces new dual feasible functions. The derived bound is able to dominate both the bounds by Martello, Pisinger and Vigo and the ones by Fekete and Schepers.

A Tabu Search algorithm for the $2D-BP$ problem was developed by Lodi, Martello, and Vigo [20]. This algorithm uses two simple construction heuristics for packing the items into the bins. The Tabu Search algorithm only controls the movement of the items between the bins. In [23], this Tabu Search approach was generalized to other variants of the BP problem, including the one considered in this paper.

Faroe, Pisinger, and Zachariasen [9] presented a Guided Local Search (GLS) heuristic algorithm for $3D-SBSBPP$. Starting with an upper bound on the number of bins obtained by a greedy heuristic, the algorithm iteratively decreases the number of bins, each time searching for a feasible packing of the boxes using the GLS method. Up to now, this heuristic is the one that obtains the best solutions for $3D-SBSBPP$.

Different constructive algorithms have been developed for different versions of the 2D problem (see [21] for a survey). Unfortunately, these approaches cannot be directly applied to the 3D case due to the growing complexity of the accommodation of the items. Two heuristics have been developed and tested for $3D-SBSBPP$ by Martello, Pisinger, and Vigo in [24]. The first algorithm, called $S-Pack$, is based on a layer building principle derived from shelf approaches used by several authors for $2D-BP$ (e.g., [6] and [4]). The second approach, called $T-MPV$, repeatedly fills a bin after the other by means of a Branch & Bound algorithm for the single container filling developed by the authors in the same paper.

Lodi, Martello, and Vigo [22] presented a new heuristic for $3D-SBSBPP$, called *Height first - Area second* (HA). The heuristic chooses the best of two possible solutions. In the first one, the items are partitioned by height into clusters and a series of layers are obtained from each cluster. Then, the layers are packed into the bins through the Branch & Bound by Martello and Toth for the $1D-BP$ problem. The second solution is obtained resorting the items by non-increasing area of their base and new layers are built. As in the first solution, the layers are packed in the bins solving a $1D-BP$ problem. According to the results, the HA heuristic is the constructive procedure that obtains the best results on benchmark tests.

Crainic, Perboli, and Tadei [7] defined the Extreme Points, an extension of the Corner Points able to better exploit the container's volume. The basic idea is to efficiently find the points where an item can be added to an existing packing. The Extreme Points are used by the authors to design new constructive heuristics based on the First Fit Decreasing and the Best Fit Decreasing heuristics for the mono-dimensional BP problem. Computational results show that they outperform all the other constructive heuristics for both $2D-SBSBPP$ and $3D-SBSBPP$. Moreover, the procedure obtains, in negligible time, results that are better than those of the Branch & Bound by Martello, Pisinger, and Vigo [24] and comparable to those of existing meta-heuristics.

3 TS^2PACK: A Two-Stage Tabu Search Heuristic for $3D - SBSBPP$

The main difference between mono-dimensional and multi-dimensional packing problems is the verification of the feasibility of the packing, i.e. given a set of items I_C assigned to a container C, determining whether an accommodation of the items inside the container exists such that the items do not overlap and the packing is compatible with the container size. Usually, both in exact and heuristic procedures, the feasibility of the packing and the evaluation of the objective function of the problem are mixed. This is particularly true for meta-heuristics and their neighborhood exploration addresses phases [9], [3].

Two observations can be made at this point. Given a container and a set of items to accommodate into according to a specified objective function:

- some decisions change from one problem to another and are usually related to the objective function, e.g., the assignment of items to bins in the BP problem or the selection of the subset of items to load in the Container Packing problem, and can be seen as optimality-related decisions;
- the problem of evaluating the accommodation is the same in different multi-dimensional packing problems and are feasibility-related decisions.

Dissociating the feasibility and the optimality issues by using two different heuristics would then provide the means to define different solution representations and apply different methods to solve the two sub-problems. Furthermore, the methods developed for the feasibility issue could be reused for different classes of packing problems and the introduction of additional constraints on the packing (e.g., guillotine cut) would not modify the heuristic dealing with the optimality issue. We thus propose TS^2PACK, a two-level meta-heuristic, where a first-level heuristic deals with the optimality of the BP problem, while a second-level heuristic finds feasible packings for the items assigned to the bins. Both heuristics are Tabu Search based.

Tabu Search is a memory-based search method introduced by Glover [14, 15]. Tabu Search meta-heuristics avoid local optima by allowing the objective function to deteriorate and series of cyclic moves in the search space by keeping track of recent moves in the so-called tabu list. The number, size, contents, and management policies of the tabu list depend on the specific problem and algorithm. The method can be enhanced by performing *diversification*, a method to guide the search toward zones of the solution space not yet explored.

The TS^2PACK meta-heuristic (see Algorithm 1 for a schema of the heuristic) computes an initial solution applying the EP-FFD heuristic developed by the authors [7]. The EP-FFD heuristic is a composite heuristic derived from the well-known First Fit Decreasing algorithm for the mono-dimensional Bin Packing problem. The items are sorted by non-increasing values of their volume and are accommodated one after the other into the existing bins. When an item cannot be loaded into an existing bin, a new bin is created. The items are accommodated into the bins by placing them on the Extreme Points. When an item k is placed in a given position (x_k, y_k, z_k) in a container, an additional item j can be accommodated in specific points, called Extreme Points (EPs). They are the orthogonal projections, on the three axes, of the points $(x_k + w_k, y_k, z_k)$, $(x_k, y_k + l_k, z_k)$ and $(x_k, y_k, z_k + h_k)$ on the items previously accommodated into the container, where w_k, l_k and h_k are width, length and depth of item k respectively.

Given an initial solution, the TS^2PACK meta-heuristic iteratively discards the bin $\bar{b}$ with the worst value of the fitness function $\varphi(b)$ defined in [23] as the weighted sum of the volume used by the items loaded into the bin and the number of items. The items in $\bar{b}$ are iteratively loaded into the other bins, selecting, for each item, the bin with the maximum value of $\varphi(b)$. More precisely, we relax the bin size constraints on the height axis. Thus, each item is accommodated on the Extreme Points into the bin that minimizes the overall height of the new packing. When the new solution is feasible, i.e. each subset of items assigned to each bin is packed such that the items do not overlap and the packing is compatible with the bin size, the new solution is accepted as the current best and a new bin is discarded. Otherwise, the first-stage Tabu Search heuristic ACC_TS is applied (see Subsection 3.1).

The ACC_TS heuristic works on the items-to-bins assignments, without taking into account explicitly the feasibility of the new solutions. It relaxes the bin dimensions and considers the unfeasibility due to building larger packings than the bin size as a penalty added to the objective function. The inner heuristic IG_TS (see Subsection 3.2) is used by the ACC_TS procedure to check the feasibility and optimize the packing in order to respect the constraints on the bin's sizes.

The TS^2PACK heuristic stops when either a given time limit is reached or the number of bins in the current solution is equal to the lower bound by Boschetti [5].

3.1 The first level ACC_TS heuristics

The ACC_TS heuristic works with a set of items and a fixed number of bins. Its goal is to find a set of items-to-bins assignments able to produce a packing for each bin such that it fits within the dimensions of the bin and the items are not overlapping. The position of the items in the bins is not determined directly by the ACC_TS heuristic. Rather, it is assigned by an external procedure which guarantees that each two items assigned to the same bin are not overlapping, but relaxes the constraints on the bin size, i.e., the boxed envelope of minimum size which contains the items accommodated into a bin can be larger than the bin itself.

ACC_TS is a Tabu Search-based heuristic. It makes use of a local-search neighborhood whose size and accuracy is dynamically varied during the search by means of the k-chain-moves procedure. It also includes a diversification phase, which is applied after a fixed number of iterations without improvement and aims to explore new regions of the solution space. At each iteration, the

Algorithm 1 The TS^2PACK meta-heuristic

```
$CS, BS$: Current and Best Solution
$it$: item
$Bins(S)$: Function returning the number of bins of a solution $S$
$LB_B$: Lower bound by Boschetti

Compute the initial solution by $EP - FFD$
$CS = BS = EPFFD$
while time limit is not reached or $BS \neq LB_B$ do
  $\bar{b} = \arg\min \varphi(b)$
  Relax the $Z$ size of the bins
  for all $it \in \bar{b}$ do
    $b' = \arg\max \varphi(b)$
    Load $it$ in the EP of $b'$ minimizing the overall height
  end for
  Discard $\bar{b}$
  if $CS$ is feasible then
    $BS = CS$
  else
    Relax the constraints on the sizes the bins
    $ACC\_TS(CS)$ (see Subsecion 3.2)
  end if

end while
```

ACC_TS heuristics considers the neighbors according to non-increasing values of their objective function and stops when the best solution found is feasible for $3D - SBSBPP$, i.e. each packing satisfies the bin dimension constraints, or when the time limit given to the overall heuristic is reached.

Solution representation

Define:

- I, the set of items;
- C, the set of bins. All bins are similar with dimensions W, D, and H;
- A system of orthogonal axes X, Y, and Z for each container with the origin in the left-back-bottom corner and parallel to the sides of the bin;
- A function $m(j) : I \longrightarrow C$ that returns the bin to which item j has been assigned.

The solution representation is then defined as follows:

- A partition of the set of the items I yielding $|C|$ subsets I_c:
  $I_c = \{j \in I : m(j) = c,\ c \in C\}$,
  $I_1 \cup ... \cup I_C = I$,
  $I_i \cap I_j = \emptyset, \forall i, j \in C, i \neq j$.
- The positions x_i, y_i, and z_i of item i on the orthogonal axes X, Y, and Z, respectively. The positions are such that the items in the same bin are not overlapping.

This solution representation does not guarantee that the packing fits into the bin.

Let

$$Inf_{cd}(S(c)) = \max\{0, L_{cd}(S(c)) - \overline{L}_{cd}\},$$

be a measure of the packing unfeasibility with respect to the size of the bin on the d^{th} axis, where $\overline{L}_{cd}$ and $L_{cd}(S(c))$ stand for the size on the d^{th} axis of the bin c and packing $S(c)$, respectively.

The following formula is then used to evaluate the unfeasibility of a packing $S(c)$ of bin c:

$$OF = lex\left(\sum_{c\in C}\sum_{d} Inf_{cd}(S(c)), \sum_{c\in C}\sum_{d} L_{cd}(S(c))\right), \qquad (1)$$

where lex is the lexicographic order of the objective functions. S is then a feasible solution of $3D - SBSBPP$ iff:

$$\sum_{c \in C} \sum_{d} Inf_{cd}\,(S(c)) = 0.$$

Because the first part of the function is "flat", i.e. several solutions have the same value, we use the second term to guide the algorithm toward solutions that are as compact as possible.

k-chain-moves procedure

Neighborhood definition is a critical issue in the design of many meta-heuristics, and Tabu Search in particular. Large neighborhoods generally conduct to better local optimal solutions and search accuracy. On the other hand, they are also increasing the time required to explore the neighborhood at each iteration and, thus, the computational burden of the method. Consequently, a larger neighborhood does not necessarily produce a more effective heuristic unless one can explore it in a very efficient manner. Thompson and Orlin [27, 28] proposed the cyclic exchange neighborhood to solve partitioning problems, an idea generalized by Ahuja, Orlin, and Sharma [2]. Ghamlouche, Crainic, and Gendreau [12, 13] proposed cycle-based neighbourhoods for Tabu Search and Path Relinking meta-heuristics addressing the fixed-cost, capacitated, multicommodity network design problem. All the above-mentioned authors define large-size neighborhoods based on the introduction of cycles of simple moves (e.g., swaps or open/close a single arc). These large neighborhoods are then implicitly explored by solving particular network flow problems on specifically built and managed graphs.

The k-chain-moves procedure is a simplified version of these approaches, which introduces chains of k changes in the solution with a negligible additional computational effort. The k-chain-moves procedure does not involve the management of an additional data structure and, in this sense, it is also somewhat related to the ejection-chain concept proposed by Glover ([16]; see also [26]) and used mainly in the context of Tabu Search.

Consider a combinatorial optimization problem P and its set of feasible solutions F. A neighborhood of a feasible solution $s \in F$ may be described as a function associating a subset of feasible solutions $N(s) \subset F$ to s, while an associated move is a rule that when applied to s yields $s' \in N(s)$. A large neighborhood defining cycles of 2 consecutive moves may then be trivially obtained by defining the same neighborhood and applying the move to each solution $s' \in N(s)$. The size of the new neighborhood is $|N(s)|^2$. Applying the same idea recursively k times, we obtain an increasingly larger neighborhood of size $|N(s)|^k$, with members which can be derived from s by means of a series - a *cycle* - of k moves. The computational time needed to explore this large neighborhood would increase rapidly with the size of k, however.

The k-chain-moves procedure introduces a cycle of at most k moves, by successively building the neighborhood of the best solution of the previous iteration and moving to the best neighbor in this neighborhood. The procedure is illustrated in Algorithm 2 and starts from a current solution CS. The neighborhood of CS is denoted $N(CS)$ and gives the first elements to the final large kN neighborhood identified as $NSet$. Let BS_1 be the best solution obtained after the first iteration (in $N(CS)$). The procedure then builds the neighborhood $N(BS_1)$ of BS_1, denoted $NSet_2$, and add its elements to $NSet$. The procedure continues extracting BS_2, the best solution in $NSet_2$, building its neighborhood $NSet_3 = N(BS_2)$ of BS_2, adding its elements to $NSet$, and so on and so forth. Iterating the procedure k times, $NSet$ will contain a set of solutions that differ from the starting solutions by cycles of up to k moves.

To illustrate $3D - SBSBPP$, consider the neighborhood which, given a starting bin and an item i inside it, builds the neighbors swapping i with the items in the other bins. Suppose, e.g., we apply this neighborhood to item 1 in bin $B1$ in the loading depicted in Figure 1 to build a cycle of length $k = 2$. The first iteration, the neighbors are all possible swaps of item 1 with items in the other bins. Assume the best swap is with item 2 in bin $B2$. Following the move, item 1 is in bin $B2$ and we apply the neighborhood with item 1 in bin $B2$ on the remaining $B - 2$ bins, obtaining the swap $1 - 3$ (neither the bin $B1$, nor the bin $B2$ is considered to avoid cycles). At the end of the procedure, we have built a solution that differs from the original one by 2 swaps.

Algorithm 2 k-chain-moves procedure

```
**Input** $k$ : length of the cycle to build
$CS$ : Current Solution
$GEN_N$ : Function generating the set of solutions of the neighborhood $N$
$AVOID$ : List of forbidden solutions
$NSet$ : Set of the solutions belonging to the neighborhood

$NSet = \{\emptyset\}$
$AVOID = AVOID \cup \{\text{CS}\}, BS_0 = CS$
for $i = 1$ to $k - 1$ do
    $NSet_i = GEN_N(AVOID, BS_{i-1})$
    $NSet = NSet \cup \{NSet_i\}$
    Extract the best solution $BS_i$ from $NSet_i$
    $AVOID = AVOID \cup \{BS_i\}$
end for
return $NSet$
```

Neighborhood structure

The structure of the neighborhood for the high-level heuristic focuses on changes of assignments of items to bins. It is a composite neighborhood which uses the 1-swap and the add-drop neighborhoods. Given a target bin b and the list of the items accommodated in it, the 1-swap neighborhood swaps each item in b with the items assigned to another bin, while the add-drop neighborhood unloads one item from b and reloads it into another bin. To increase the quality of the

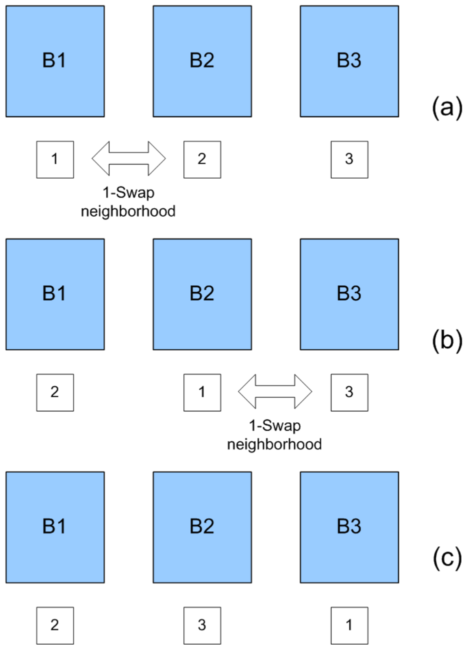

Figure 1: Example of 2-chain-moves application

composite neighborhood, we use the 1-swap neighborhood within the k-chain-moves procedure.

The procedure that builds this neighborhood is described in Algorithm 3. The target bin $\overline{b}$ is the bin with the worst contribution to the objective function. Then the k-chain-moves is applied with the 1-swap neighborhood. Finally, the add-drop neighborhood is applied. The packing of bins in the candidate solution which are not feasible, i.e. the packing does not respect the bin dimensions, is optimized by means of the IG_TS heuristic presented at the next sub-section.

It is simple to see that the 1-swap neighborhood has size $O(|I_{\overline{b}}|n)$, where $|I_{\overline{b}}|$ is the number of items loaded in bin $\overline{b}$, while the add-drop neighborhood is $O(C|I_{\overline{b}}|)$, where C is the number of bins in the solution. Thus, the composite neighborhood is $O(|I_{\overline{b}}|kn + C|I_{\overline{b}}|)$.

Tabu list structure

The tabu list records the last assignments of an item to a bin. This prevents the reversal of the assignment status as long as they remain in the list. Given a solution in the composite neighborhood, the tabu moves are built as follows:

1. If we swap item i in bin l with item j in bin m, we forbid the assignment of i to l and j to m;

2. If we move item i from bin l to bin m, we forbid the assignment of i to l.

Algorithm 3 ACC_TS's composite-neighborhood

```
Input $K$: Length of the chain to build
Input $CS$: Current solution

$AD(s,b)$: Add and Drop neighborhood of solution $s$ and target bin $b$
$SW(s,b,k)$: 1-Swap neighborhood of solution $s$ and target bin $b$ with $k$-chain-moves procedure with size $k$
$N(i)$: Whole neighborhood of solution $s$ and target bin $b$
$BS$: Best solution

$\bar{b} = \arg\max lex\{\sum_d Inf_{cd}(S), \sum_d L_{cd}(S)\}$
$N(CS) = \{\emptyset\}$
Generate $SW(BS, \bar{b}, k)$
Extract the best solution from $SW(CS)$ and assign it to $BS$
$\bar{b} = \arg\max lex\{\sum_d Inf_{cd}(S), \sum_d L_{cd}(S)\}$
Generate $AD(BS, \bar{b})$
$N(CS) = SW(CS) \cup AD(BS, \bar{b})$
```

The tabu list has a fixed length.

Diversification phase

The diversification phase is applied once a fixed number of non improving iterations is reached. Given C as the number of bins in the current solution, the diversification phase changes the current solution of the ACC_TS heuristic by performing on the first $C/2$ bins in the solution the k-chain-moves procedure (with $k = C/2$) applied to the add-drop neighborhood, and taking the best solution of the resulting neighborhood as the new solution.

3.2 Feasibility of a packing: the IG_TS procedure

The aim of the second-level procedure is to find the accommodation of the items that minimizes the unfeasibility of the resulting packing as defined in Equation (1). The IG_TS procedure is a Tabu Search-based local search, which uses the implicit solution representation given by the Interval Graph (IG) approach proposed by Fekete and Schepers [10, 11] and thus reduces the search space.

Given a list of items to be accommodated into a bin, the procedure first builds an initial packing by means of an Extreme Point-based heuristic derived from [7]. This heuristics relaxes the height of the bin and considers the items one after the other without any sorting. Each item is loaded on the EP minimizing the Z, Y, and X coordinates, in this order. The resulting packing is transformed to yield the representation by Fekete and Schepers and the Tabu Search-based local search is applied. The heuristic works directly on the IGs, and uses a fixed sized tabu-list (no long-term memories or diversification phase are used). At procedure considers the neighbors according to non-increasing values of their objective function and stops when the number of the iterations is reached, or when the best solution defines a packing compatible with the bin dimensions.

Solution Representation

According to the representation of Fekete and Schepers, a three-dimensional packing and the projections of one of its items i on the d^{th} coordinate axis generate intervals on the real numbers set, which can be described using a graph-based representation. We recall in this sub-section a number of basic definitions. For a more detailed description of this representation, refer to [10, 11].

Consider the two packings in Figure 2. They are made by the same set of items, use the same volume, and are characterized by the same rectangular minimal envelope. One notices a common combinatorial structure, the differences arising from how items "see" each other orthogonally to the frame axes. Projecting the items on the axes (see Figure 2), one may observe how the item projections overlap on each axis. Associate to each axis a graph $G_d(V, E_d)$, where the nodes represent the items and two nodes are connected by an edge if the corresponding projections on the d^{th} axis overlap (even partially). The graphs associated to the two packings are the same. Moreover, by construction, the graphs are *Interval Graphs* (*IG*s).

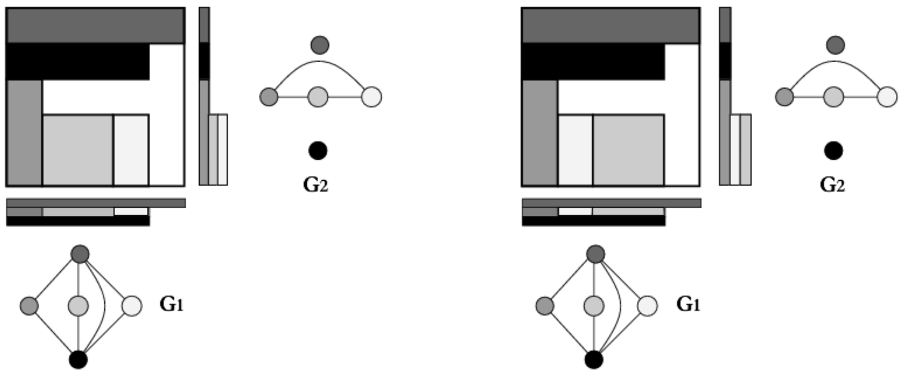

Figure 2: Example of definition of packings by the definition by Fekete and Schepers

Given an *IG*, its co-graph is a *Comparability Graph* (*CG*). A transitive orientation of the *CG* determines the positions of the items on the axis. Thus, we may build a graph representation for a 3D packing by computing, for each coordinate axis, its *IG* representation and vice versa. Moreover, a set of *IG*s defines not a single packing, but a set of packings with the same minimal boxed envelope. For instance, the *IG*s in Figure 2 represent 36 different packings. Define:

- C, a container with width W, depth D, and height H;

- I_C, the set of items assigned to a container;
- A system of orthogonal axes X, Y and Z with origin O in the left-back-down corner of the container and parallel to the sides of the bin;
- The positions x_i, y_i, and z_i of the left-back-down corner of item $i \in I_C$ in the packing, on the orthogonal axes X, Y and Z.

Then, more formally, a set of 3 graphs G_d^C, on the three axes d, with vertex set $V = I_C$ and respective edge sets E_d, are defined as a *Packing Class* (PC) in 3 dimensions for container C if:

1. Each G_d^C is an IG;
2. $\forall i, j \in V$, it exists a dimension d such that $e_{ij} \notin E_d$,;
3. Each stable set of G_d^C is $d - feasible$, i.e. the items associated to the stable sets can be lined up on the axis d, $\forall d$, without exceeding the size of the bin on the same axis.

Conditions 1 and 3 can be efficiently checked by exploiting the properties of the IGs, the transitive orientations of the co-graphs of G_i^C defining the packings in the PC. Moreover, all the packings in a PC have the same box envelope. An efficient way to check condition 1 then is to verify whether the graph does not contain chordless cycles with 4 nodes and its co-graph is a CG. In the following, we will refer to a chordless cycle with 4 nodes as a $C4$ chordless cycle. The second task can be accomplished by means of the *Transitively Orientable* (TRO) algorithm by Golumbic [17], which verifies whether a graph is a CG by building one of its transitive orientation in $O(\delta|E|)$, where δ is the maximum degree of a vertex of the graph and $|E|$ is the number of edges (see [17] for details on IGs, transitive orientations, and the TRO algorithm).

When condition 3 holds, the envelope of the packings represented by a PC lies inside the container C. We represent the solutions by means of a relaxation of the PCs, where conditions 1 and 2 hold, while condition 3 is ignored. We call a solution satisfying conditions 1 and 2 a *Semi-Packing Class* (SPC). Similarly to the PC, each SPC defines a set of actual packings characterized by the same minimal boxed envelope.

We thus define the feasible solutions and the search space of the IG_TS heuristic as the set of the SPCs that can be built with the set I_C. Given a SPC, called s, we define $L(s, d)$ as the size of the packings belonging to s on the d axis (all the packings have the same minimal boxed envelope). Each solution is evaluated by means of the objective function (1). According to this definition, a SPC is a PC iff $Inf_{cd}(s) = 0$, $\forall d = \{X, Y, Z\}$.

Neighborhood structure

The neighborhood we propose for a given SPC is defined by changes in item overlapping. Recall that the overlapping of two items i and j in dimension d

	X	**Y**	**Z**	G_X^C	G_Y^C	G_Z^C
1	O	O	NO	$e_{ij} \in E_X$	$e_{ij} \in E_Y$	$e_{ij} \notin E_Z$
2	O	NO	O	$e_{ij} \in E_X$	$e_{ij} \notin E_Y$	$e_{ij} \in E_Z$
3	NO	O	O	$e_{ij} \notin E_X$	$e_{ij} \in E_Y$	$e_{ij} \in E_Z$
4	O	NO	NO	$e_{ij} \in E_X$	$e_{ij} \notin E_Y$	$e_{ij} \notin E_Z$
5	NO	O	NO	$e_{ij} \notin E_X$	$e_{ij} \in E_Y$	$e_{ij} \notin E_Z$
6	NO	NO	O	$e_{ij} \notin E_X$	$e_{ij} \notin E_Y$	$e_{ij} \in E_Z$
7	NO	NO	NO	$e_{ij} \notin E_X$	$e_{ij} \notin E_Y$	$e_{ij} \notin E_Z$

Table 1: Overlapping rules for two items i and j in three dimensions (O = Overlapping, NO = Not Overlapping)

corresponds to adding the edge e_{ij} to the IGs G_d^C of the SPC. Moreover, we can evaluate the size of the packings represented by the SPC. We therefore define the neighborhood directly on the graphs G_d^C.

The neighborhood is denoted *overlapping rule exchange* and is displayed as Algorithm 4. It is defined by considering each pair of items (i, j) in the container and changing the edges in the three IGs describing the corresponding packing. Seven such feasible combinations can be obtained, as shown in Table 1, each defining an *overlapping rule* between items i and j.

Algorithm 4 Overlapping rule exchange

```
**Input** $CS$: current $SPC$ defined by the $IG$s $G_x^C(V_x, E_x)$, $G_y^C(V_y, E_y)$, and $G_z^C(V_z, E_z)$

**for all** item $i$ **do**
  **for all** item $j \neq i$ **do**
    **for all** overlapping rule $r$ **do**
      Apply the overlapping rule
      **for all** dimensions d $\in \{X, Y, Z\}$ **do**
        **if** $r$ adds the edge $e_{ij}$ to $E_d$ **then**
          $E_d = E_d \cup \{e_{ij}\} \cup \{e_{ik} : p_k \leq p_j, p_k + l_k \geq p_j + l_j, p_k, p_j \in P_d, l_k, l_j \in L_d\}$
        **else**
          $E_d = E_d \setminus \{\{e_{ij}\} \cup \{e_{ik} : p_k \geq p_l, p_k + l_k \leq p_j + l_j, p_k, p_j \in P_d, l_k, l_j \in L_d\}\}$
        **end if**
      **end for**
      **if** $G_d^C(V_d, E_d\prime)$ are $IG$s, $\forall d$ **then**
        Add the solution to the neighborhood
      **end if**
    **end for**
  **end for**
**end for**
```

Recall that, by the properties of the IGs, a SPC represents a group of accommodations of items, each packing displaying the same overlapping between the items. Then, under particular conditions, a new SPC representing the same packing may be obtained by adding or removing a single edge from an IG of a SPC, as illustrated in the following. Consider the packing in Figure 3a. The IG related to the Y axis is represented by the graph depicted in Figure 3b. Let us modify this IG by adding the edge $(3, 4)$ on Y, leaving the IG on X unchanged.

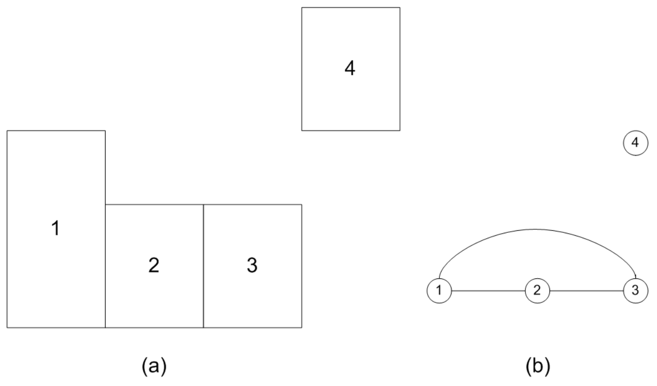

Figure 3: Example of the add overlapping rule

Then we obtain, on the Y axis, the IG in Figure 4b. Its co-graph is represented in Figure 4c. To obtain one of the accommodations of the items represented by the SPC, we build the transitive orientation of the co-graph represented in Figure 4d. Assigning the positions of the items on the Y dimension according to the transitive orientation, we obtain the packing of Figure 4a, which is the same one represented in Figure 3a. Thus, the same packing can belong to different SPCs. One can easily check that the same situation is obtained when edges $(2, 4)$ and $(1, 4)$ are added.

Let us now consider the situation in Figure 5. Starting from the packing in Figure 3a, we add the edge $(1, 4)$ to the IG represented in Figure 3b and obtain the new graph of Figure 5b. Following the same procedure as in the previous example, the co-graph of Figure 5b is depicted in Figure 5c, while Figure 5d represents the transitive orientation of the co-graph. In this case, after assigning the positions of the items according to this transitive orientation, the accommodation of the items is changed, see Figure 5a.

These different behaviors are explained by the structure of the IGs involved and the position of the corresponding items. Thus, items $1, 2$ and 3 form a clique in the IG and, thus, each pair is (partially, in case) overlapping on the Y axis. Furthermore, the positions of the items in Figure 3a have the same y value, even though the height of item 1 differs from the heights of items 2 and 3. ¿From a logical point of view, in fact, the first example corresponds to using the following add overlapping rule: *"Items 3 and 4 overlap, but item* 4 *does not overlap with items 1 and 2"*. This requirement cannot be satisfied, however, because the overlapping of items 3 and 4 implies to add the overlapping with the other two items too. This situation is managed by the IG on the Y axis

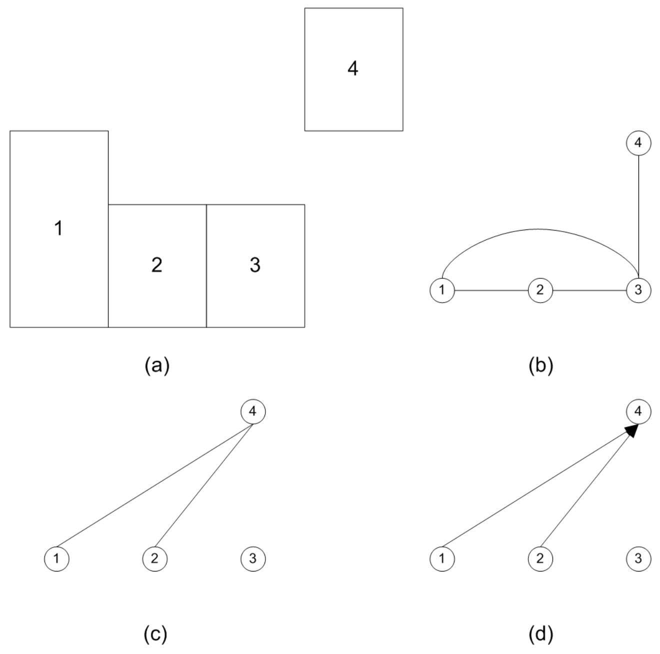

Figure 4: Adding edge $(3,4)$ to an IG

with the co-graph, which does not change the represented packing.

A heuristic rule is used to reduce the probability of cycling when changing the SPCs by adding or removing edges. The heuristic proceeds by considering the edge one wants to add or drop as well as the other edges sharing a node and displaying the properties indicated in the following. Let $G_d = (V_d, E_d)$ be an IG and, for the items in V_d, let P_d and L_d represent the sets of item positions and lengths on axis d, respectively. To add an edge e_{ij} to E_d, we then build the new graph G_2, which defines the add overlapping rule (move):

$$\begin{aligned} & G_2 = (V_d, E'_d) \\ & E'_d = E_d \cup \{e_{ij}\} \cup \{e_{ik} : p_k \leq p_j, p_k + l_k \geq p_j + l_j, p_k, p_j \in P_d, l_k, l_j \in L_d\}\,. \end{aligned} \tag{2}$$

Thus, according to the overlapping rule (2), adding an edge e_{ij} to an existing IG of a SPC implies that one also adds the edges e_{ik} related to the items k that have their end points higher than item j and their left-back-down corner position at least equal to j on the considered axis. A similar remove overlapping rule (move) is defined to specify that, when removing edge e_{ij} of an existing IG of a SPC, one also removes edges e_{ik} related to the items k with the same end point on the considered axis as item j. More formally (see rule (3)), given an IG $G_d = (V_d, E_d)$ and an edge e_{ij} to be removed, the following new graph G_3

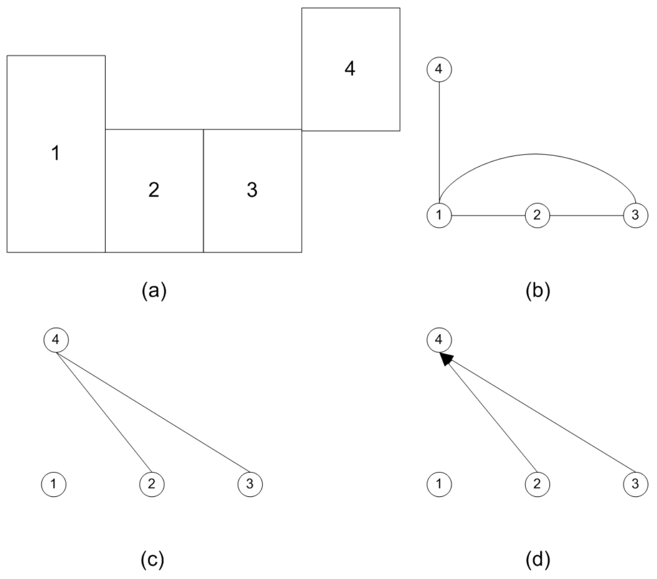

Figure 5: Adding edge $(1, 4)$ to an IG

defines the remove overlapping rule (move):

$$G_3 = (V_d, E'_d) \tag{3}$$
$$E'_d = E_d \backslash \{\{e_{ij}\} \cup \{e_{ik} : p_k \geq p_j, p_k + l_k \leq p_j + l_j, p_k, p_j \in P_d, l_k, l_j \in L_d\}\}.$$

Once the new graphs are generated, one must verify whether they are IGs. It is in fact easy to see that adding or removing an edge form an IG may yield a graph which is not an IG, as illustrated in Figure 6. The graph in Figure 6a has no $C4$ chordless cycle and is an IG with a CG as its co-graph (illustrated in Figure 6b). Removing the edge $(2, 4)$ yields the graph depicted in Figure 6c. This graph is without $C4$ chordless cycles, but its co-graph (Figure 6d) is not a CG and, thus, the corresponding graph is not an IG. A similar example can be build for the case of adding an edge to an existing IG.

If the new graphs are IGs, the objective function is evaluated and the new solution is inserted in the neighborhood, otherwise the solution is discarded. The size of the neighborhood is $O\left(|I_C|^2\right)$. Moreover, both the verification whether a new graph is an IG and the computation of the transitive orientations can be efficiently performed by means of the properties of IGs (see [17]).

Tabu list structure

Moves belong to the tabu list each time a solution is selected as follows. Suppose the overlapping rule yielding the selected solution is j. A new solution will be considered "tabu" if its overlapping rule is j or $\overline{j}$, where $\overline{j}$ is the move that is obtained by substituting, on each dimension, the overlapping rule j with its

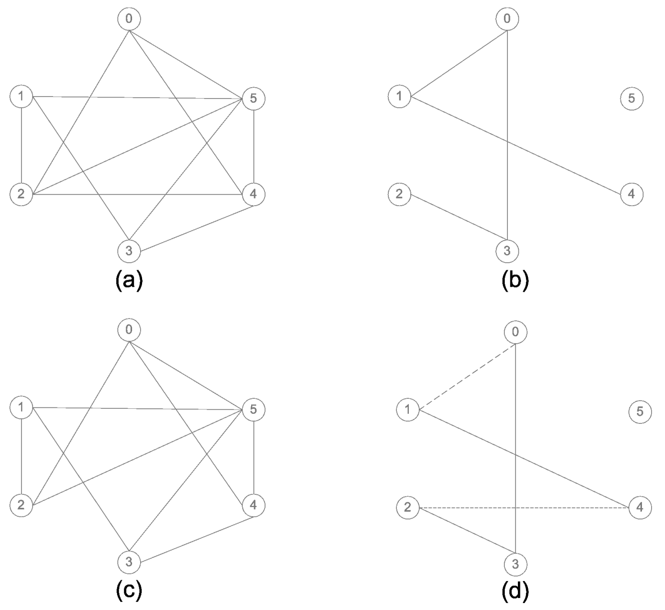

Figure 6: Example of removing an edge from an IG

inverse. For instance, if j is the first overlapping rule in Table 1, its reverse move $\bar{j}$ is the overlapping rule 6. The tabu list has a fixed length.

4 Computational results

In this section we present and analyze the results of a rather comprehensive computational experimentation. The performance of the algorithm is compared with that of the exact algorithm for $3D-SBSBPP$ by Martello, Pisinger, and Vigo [24] and the GLS by Faroe, Pisinger, and Zachariasen [9]. We do not report the results of the Tabu Search algorithm by Lodi, Martello, and Vigo [20] because their procedure is outperformed by GLS.

$3D-SBSBPP$ instances come from [24]. For Classes 1 to 3, 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 uniformly random in $[\frac{2}{3}H, H]$, d_j uniformly random in $[\frac{2}{3}D, D]$;
- *Type 2*: w_j uniformly random in $[\frac{2}{3}W, W]$, h_j uniformly random in $[1, \frac{1}{2}H]$, d_j uniformly random in $[\frac{2}{3}D, D]$;
- *Type 3*: w_j uniformly random in $[\frac{2}{3}W, W]$, h_j uniformly random in $[\frac{2}{3}H, H]$, d_j uniformly random in $[1, \frac{1}{2}D]$;
- *Type 4*: w_j uniformly random in $[\frac{1}{2}W, W]$, h_j uniformly random in $[\frac{1}{2}H, H]$, d_j uniformly random in $[\frac{1}{2}D, D]$;

- *Type 5*: w_j uniformly random in $[1, \frac{1}{2}W]$, h_j uniformly random in $[1, \frac{1}{2}H]$, d_j uniformly random in $[1, \frac{1}{2}D]$.

The classes are built as follows:

- *Class 1*: type 1 with probability 60%, type 2, 3, 4, 5 with probability 10% each;
- *Class 2*: type 4 with probability 60%, type 1, 2, 3, 5 with probability 10% each;
- *Class 3*: type 5 with probability 60%, type 1, 2, 3, 4 with probability 10% each.

Classes from 4 to 6 are generated according to the rules by Berkey–Wang [4]:

- *Class 4*: w_j, h_j and d_j uniformly random in [1,10] and $W = H = D = 10$;
- *Class 5*: w_j, h_j and d_j uniformly random in [1,35] and $W = H = D = 40$;
- *Class 6*: w_j, h_j and d_j uniformly random in [1,100] and $W = H = D = 100$.

For each class (i.e., 1, 2, 3, 4, 5, and 6) we consider instances with a number of items equal to 50, 100, 150, and 200. Given a class and an instance size, we generate 10 different problem instances based on different random seeds. Bins are cubic in all instances.

The results of the benchmark algorithms GLS and MPV are taken from [24] and [9], respectively. Both algorithms are run on a Digital 500 workstation with a 500 MHz 21164 CPU with a time limit of 1000 seconds for each instance. TS^2PACK is coded in C++ and tested on a Pentium4 2000 Mhz CPU, with a time limit of 300 seconds to solve each instance. We used the results of the SPEC CPU2000 benchmarks published in [1] to obtain the equivalence in terms of performance between the Digital and the Pentium4 computers. In the following, when citing computational times, we refer to the equivalent computational time on the Digital 500 workstation, i.e. if the computer in use is not a Digital 500 workstation the computational time is changed by a ratio given by the SPEC CPU2000 benchmarks.

In the IG_TS procedure, the iteration limit is fixed to the number of the items assigned to the bin, while the size of the Tabu List is equal to the minimum between the items assigned to the bin and 7.

First, we present the tuning of the parameters of the heuristic, including the size of the cycles of moves induced by the k-chain-moves procedure in Subsection 4.1. Section 4.2 is then devoted to compare the TS^2PACK heuristic to state-of-the-art methods.

Class	Bins	n	k=1	k=2	k=3
1	100x100	50	13.4	13.4	13.4
		100	26.7	26.7	26.7
		150	37.2	37	37
		200	51.3	51.2	51.1
2	100x100	50	29.4	29.4	29.4
		100	58.9	58.9	58.9
		150	86.8	86.8	86.8
		200	118.8	118.8	118.8
3	100x100	50	8.3	8.3	8.3
		100	15.6	15.2	15.2
		150	20.6	20.3	20.1
		200	27.7	27.5	27.4
4	10x10	50	9.8	9.8	9.8
		100	19.2	19.1	19.1
		150	29.3	29.3	29.2
		200	37.8	37.8	37.7
5	40x40	50	7.4	7.4	7.4
		100	12.3	12.3	12.3
		150	15.9	15.8	15.8
		200	23.8	23.7	23.5
6	100x100	50	9.3	9.2	9.2
		100	18.9	18.8	18.8
		150	25.1	24.9	24.8
		200	30.5	30.3	30.3
Total bins			734	731.9	731

Table 2: TS^2PACK performance for various k parameter values in the k-chain-moves procedure

						GAP WITH LB	
Class	Bins	n	IG_TS	EP-FFD	LB	IG_TS	EP-FFD
1	100x100	50	13.4	13.7	12.9	3.88%	6.20%
		100	26.7	27	25.6	4.30%	5.47%
		150	37	37.4	35.8	3.35%	4.47%
		200	51.1	53	49.7	2.82%	6.64%
2	100x100	50	29.4	29.4	29	1.38%	1.38%
		100	58.9	59	58.5	0.68%	0.85%
		150	86.8	86.8	86.4	0.46%	0.46%
		200	118.8	118.8	118.3	0.42%	0.42%
3	100x100	50	8.3	8.3	7.6	9.21%	9.21%
		100	15.2	15.6	14	8.57%	11.43%
		150	20.1	21.4	18.8	6.91%	13.83%
		200	27.4	28.3	26	5.38%	8.85%
4	10x10	50	9.8	9.8	9.4	4.26%	4.26%
		100	19.1	19.1	18.4	3.80%	3.80%
		150	29.2	29.3	28.5	2.46%	2.81%
		200	37.7	38	36.7	2.72%	3.54%
5	40x40	50	7.4	7.4	6.8	8.82%	8.82%
		100	12.3	12.4	11.5	6.96%	7.83%
		150	15.8	16.2	14.4	9.72%	12.50%
		200	23.5	24.1	22.7	3.52%	6.17%
6	100x100	50	9.2	9.3	8.7	5.75%	6.90%
		100	18.8	18.9	18.4	2.17%	2.72%
		150	23.9	24.2	22.5	6.22%	7.56%
		200	30	30.2	28.2	6.38%	7.09%
Total bins			729.8	737.6	708.8	2.96%	4.06%

Table 3: TS^2PACK performance with and without the IG_TS procedure

4.1 Algorithm tuning

The parameter values affecting the computational behavior of the overall algorithm are the number of iterations of the IG_TS heuristic and the length of the cycles in the k-chain-moves heuristic. It is intuitive that the computational time of each iteration of the ACC_TS heuristic increases proportionally with the number of iterations of the IG_TS heuristic. According to our results, performing more than 7 iterations of the IG_TS heuristic hurts the performances of the overall heuristic. Moreover, to reduce computational time, we apply the IG_TS heuristic not to all the packings in the macro-neighborhood of the ACC_TS heuristic, but only to its candidate solution.

The results of the impact of k-chain-moves heuristic are summarized in Table 2. The class of the instances, the sizes on the bins, and the number of items are reported in the first three columns, respectively. The other columns show the mean number of bins over 10 instances when the size of the k parameter of the k-chain-moves procedure is fixed to 1, 2, and 3, respectively. We do not report the results for values more than 3, because they does not produce better solutions due to the computation effort of the resulting procedure. Notice that $k = 1$ corresponds to not applying the k-chain-moves procedure, while $k = 2$ and $k = 3$ imply the introduction of cycles of item assignment to bins of length 2 and 3, respectively. One notices that, on small-sized instances, the k-chain-moves procedure is not relevant. The effects of the procedure can be noticed when the number of the items involved increases. In particular, the impact of the k-chain-moves procedure is higher on class 3, which includes instances with items significantly smaller than the bins.

The other parameter values of the ACC_TS and IG_TS heuristics were determined experimentally considering, for each combination of class and number of items, two instances only and reducing the computational effort to 20 seconds. We noticed a generally small sensitivity of the results with respect to changes in the parameter values.

Table 3 displays the results of the TS^2PACK algorithm when the IG_TS heuristic is applied (column IG_TS) and when the IG_TS heuristic is replaced by its initial solution only (column EP_FFD), comparing them with the results of the lower bound by Boschetti [5]. The mean gap between the two versions is about 1%, which indicates that the IG_TS heuristic is effective. In particular, it will contribute to achieve state-of-the-art results using TS^2PACK, as shown in the next section.

4.2 TS^2PACK computational results

The comparison between the performances of TS^2PACK, GLS heuristic [9] and MPV, the Branch & Bound algorithm by Martello, Pisinger, and Vigo [24], is presented in Table 4. The class of the instances, the sizes on the bins, and the number of items are reported in the first three columns, respectively. We report in the next three columns, the mean number of bins over 10 instances obtained by TS^2PACK, GLS and MPV, stopping them after 1000 seconds for

Class	Bins	n	TS^2PACK	GLS	MPV	LB
1	100x100	50	13.4	13.4	13.5	12.9
		100	26.7	26.7	27.3	25.6
		150	37	37	38.2	35.8
		200	**51.1**	51.2	52.3	49.7
2	100x100	50	29.4	29.4	29.4	29
		100	**58.9**	59	59.1	58.5
		150	86.8	86.8	87.2	86.4
		200	**118.8**	119	119.5	118.3
3	100x100	50	8.3	8.3	9.1	7.6
		100	15.2	**15.1**	17.5	14
		150	**20.1**	20.2	24	18.8
		200	27.4	**27.2**	31.8	26
4	10x10	50	9.8	9.8	9.8	9.4
		100	19.1	19.1	19.4	18.4
		150	**29.2**	29.4	29.6	28.5
		200	37.7	37.7	38.2	36.7
5	40x40	50	7.4	7.4	8.1	6.8
		100	12.3	12.3	15.3	11.5
		150	15.8	15.8	19.7	14.4
		200	23.5	23.5	27.9	22.7
6	100x100	50	9.2	9.2	10.1	8.7
		100	**18.8**	18.9	20.2	18.4
		150	23.9	23.9	27.3	22.5
		200	30	**29.9**	34.9	28.2
Total bins			729.8	730.2	769.4	708.8

Table 4: Comparing TS^2PACK, GLS, and MPV

Class	Bins	n	TS^2PACK	GLS	TS^2PACK	GLS	TS^2PACK	GLS
			60 s		150 s		1000 s	
1	100x100	50	13.4	13.4	13.4	13.4	13.4	13.4
		100	27	26.9	26.7	26.7	26.7	26.7
		150	37.7	37.5	37	37.2	37	37
		200	53	52.8	51.1	52.1	51.1	51.2
2	100x100	50	29.4	29.4	29.4	29.4	29.4	29.4
		100	59.2	59	58.9	59	58.9	59
		150	87.3	87.1	86.8	86.9	86.8	86.8
		200	119.2	119.9	118.8	119.7	118.8	119
3	100x100	50	8.3	8.3	8.4	8.3	8.3	8.3
		100	15.4	15.1	15.3	15.1	15.2	15.1
		150	20.9	20.7	20.5	20.3	20.1	20.2
		200	28	27.8	27.6	27.5	27.4	27.2
4	10x10	50	9.9	9.8	9.8	9.8	9.8	9.8
		100	19.5	19.3	19.1	19.1	19.1	19.1
		150	29.4	29.5	29.2	29.4	29.2	29.4
		200	38.7	38.5	37.7	38	37.7	37.7
5	40x40	50	7.4	7.4	7.4	7.4	7.4	7.4
		100	12.3	12.3	12.3	12.3	12.3	12.3
		150	16	15.8	15.8	15.8	15.8	15.8
		200	24.8	24.4	23.5	24.1	23.5	23.5
6	100x100	50	9.2	9.2	9.2	9.2	9.2	9.2
		100	19.2	18.9	18.8	18.9	18.8	18.9
		150	24.6	24.5	24.8	24.1	23.9	23.9
		200	30.8	30.6	30	30.1	30	29.9
Total bins			740.6	738.1	731.5	733.8	729.8	730.2

Table 5: Comparing TS^2PACK and GLS after 60, 150 and 1000 seconds

each instance. Notice that the results of MPV, being the algorithm stopped after a fixed amount of time, are not the proved optimal values. For GLS the results displayed are taken from the literature, while for MPV the results have been obtained by the code available from the authors, modified according to the erratum by Boef [8]. The algorithm run on a Pentium4 2000 Mhz CPU, with a time limit of 1000 seconds to solve each instance. The last column reports the mean number of bins over 10 instances obtained by means of the lower bound by Boschetti [5].

The figures show that TS^2PACK obtained better results than both GLS and MPV. Usually, GLS and TS^2PACK find the same results on small instances, while the gap increases with the size of the instance. In general, our algorithm achieves the same or better results than GLS. The most difficult instances for TS^2PACK are in class 3, characterized by items with a volume that is quite smaller than that of the bin. In those instances, TS^2PACK spends a lot of time in each iteration of the IG_TS heuristic, even when, after discarding a new bin, the solutions in the neighborhood are probably not feasible for $3D - SBSBPP$. The solution quality may be slightly improved increasing the maximum time of the heuristic.

The results show a total gap between TS^2PACK and GLS heuristics less than 1%. Table 5 compares the behavior of the two heuristics after 60, 150, and 1000 seconds. In general, GLS obtains better results after 60 seconds, while the opposite is true after 150 seconds, where the total gap between TS^2PACK and GLS is of 2.3 bins. The gap reduces when the time limit increases, even if TS^2PACK continues to be the algorithm with the best overall behavior. This is probably due to the fact that both algorithms are able to achieve optimal solutions for a good percentage of the instances. We tried to verify if the solution of the TS^2PACK could be improved by using them as initial solutions of MPV truncated after 10000 seconds, but for three instances only, an improvement of one bin was observed.

5 Conclusions

In this paper, we presented a new tabu search-based two-level approach for $3D - SBSBPP$. This approach separates the search for the optimal number of bins from the optimization of the accommodation of items within bins, resulting into a more flexible procedure than the existing ones. Within this framework, we extended the Interval Graph representation of packings by Fekete and Schepers to make it usable within a heuristic framework. We also introduced the k-chain-moves procedure, a general method to dynamically increase the size of a neighborhood and the quality of the associated solution, without significantly increasing the computational burden. Extensive computational results on benchmark problem instances show that TS^2PACK outperforms other methods for $3D - SBSBPP$, obtaining very good results when short computation times are available.

6 Acknowledgements

Partial funding for this project has been provided by the Natural Sciences and Engineering Council of Canada. This research has been partially supported by ASI, the Italian Space Agency, under the ICARO Project, n. ASI I/R/137/01.

References

[1] Standard performance evaluation corporation CPU2000.
URL `http://www.spec.org/cpu2000/results/cpu2000.html`

[2] R. Ahuja, J. Orlin, D. Sharma, Very large-scale neighborhood search, International Transactions in Operational Research 7 (2000) 301–317.

[3] R. Alvarez-Valdes, F. Parreno, J. Tamarit, A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems, Journal of the Operational Research Society 4 (56) (2005) 414–425.

[4] J. O. Berkey, P. Y. Wang, Two dimensional finite bin packing algorithms, Journal of the Operational Research Society 38 (1987) 423–429.

[5] M. A. Boschetti, New lower bounds for the finite three-dimensional bin packing problem, Discrete Applied Mathematics 140 (2004) 241–258.

[6] F. K. R. Chung, M. R. Garey, D. S. Johnson, On packing two-dimensional bins, SIAM - Journal of Algebraic and Discrete Methods 3 (1) (1982) 66–76.

[7] T. G. Crainic, G. Perboli, R. Tadei, Extreme point-based heuristics for three-dimensional bin packing, INFORMS Journal on Computing forthcoming.

[8] E. den Boef, J. Korst, S. Martello, D. Pisinger, D. Vigo, Erratum to "the three-dimensional bin packing problem": Robot-packable and orthogonal variants of packing problems, Operations Research 53 (4) (2005) 735–736.

[9] O. Faroe, D. Pisinger, M. Zachariasen, Guided local search for the three-dimensional bin packing problem, INFORMS Journal on Computing 15 (3) (2003) 267–283.

[10] S. P. Fekete, J. Schepers, A new exact algorithm for general orthogonal d-dimensional knapsack problems, ESA '97, Springer Lecture Notes in Computer Science 1284 (1997) 144–156.

[11] S. P. Fekete, J. Schepers, A combinatorial characterization of higher-dimensional orthogonal packing., Math. Oper. Res. 29 (2) (2004) 353–368, doi http://dx.doi.org/10.1287/moor.1030.0079.

[12] I. Ghamlouche, T. G. Crainic, M. Gendreau, Cycle-based Neighbourhoods for Fixed-Charge Capacitated Multicommodity Network Design, Operations Research 51 (4) (2003) 655–667.

[13] I. Ghamlouche, T. G. Crainic, M. Gendreau, Path Relinking, Cycle-based Neighbourhoods and Capacitated Multicommodity Network Design, Annals of Operations Research 131 (2004) 109–133.

[14] F. Glover, Tabu search - part I, ORSA Journal of Computing 1 (3) (1989) 190–206.

[15] F. Glover, Tabu search - part II, ORSA Journal of Computing 2 (1) (1990) 4–32.

[16] F. Glover, Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems, Discrete Applied Mathematics 49 (1992) 231–255.

[17] M. C. Golumbric, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, USA, 1980.

[18] D. S. Johnson, Near-optimal bin packing algorithms, Ph.D. thesis, Dept. of Mathematics, M.I.T., Cambridge, MA (1973).

[19] A. Lodi, S. Martello, M. Monaci, Two-dimensional packing problems: a survey, European Journal of Operational Research 141 (2002) 241–252.

[20] A. Lodi, S. Martello, D. Vigo, Approximation algorithms for the oriented two-dimensional bin packing problem, European Journal of Operational Research 112 (1999) 158–166.

[21] A. Lodi, S. Martello, D. Vigo, Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, INFORMS Journal on Computing 11 (1999) 345–357.

[22] A. Lodi, S. Martello, D. Vigo, Heuristic algorithms for the three-dimensional bin packing problem, European Journal of Operational Research 141 (2002) 410–420.

[23] A. Lodi, S. Martello, D. Vigo, Tspack: A unified tabu search code for multi-dimensional bin packing problems, Annals of Operations Research 131 (2004) 203–213.

[24] S. Martello, D. Pisinger, D. Vigo, The three-dimensional bin packing problem, Operations Research 48 (2) (2000) 256–267.

[25] G. Perboli, Bounds and heuristics for the packing problems, Ph.D. thesis, Politecnico di Torino, available at http://www.orgroup.polito.it/People/perboli/phd-thesys.pdf (2002).

[26] C. Rego, C. Roucairol, A Parallel Tabu Search Algorithm Using Ejection Chains for the VRP, in: I. Osman, J. Kelly (Eds.), Meta-Heuristics: Theory & Applications, Kluwer Academic Publishers, Norwell, MA, 1996 253–295.

[27] P. Thompson, J. Orlin, The theory of cyclic transfers, Tech. rep., Operations Research Center, MIT, Cambridge, MA (1989).

[28] P. Thompson, H. Psaraftis, Cyclic transfer algorithms for multivehicle routing and scheduling problems, Operations Research 41 (1993) 935–946.

[29] G. Wäscher, H. Haussner, H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research forthcoming, doi http://dx.doi.org/10.1016/j.ejor.2005.12.047.