Noname manuscript No.
(will be inserted by the editor)

Local Search for a Multi-Drop Multi-Container Loading Problem

Sara Ceschia · Andrea Schaerf

Abstract We consider a complex variant of the *Container Loading Problem*, arising from a real-world industrial application. It includes several features such as multiple containers, box rotation, and bearable weight, which are of importance in many practical situations. In addition, it also considers the situation in which boxes have to be delivered to different destinations (*multi-drop*).

Our solution technique is based on local search metaheuristics. Local search works on the space of sequences of boxes to be loaded, while the actual load is obtained by invoking, at each iteration, a specialized procedure called *loader*. The loader inserts the boxes in the container using a deterministic heuristic which produces a load that is feasible according to the constraints.

We test our solver on real-world instances provided by our industrial partner, showing a clear improvement on the previous heuristic solution. In addition, we compare our solver on benchmarks from the literature on the basic container loading problems. The outcome is that the results are in some cases in-line with the best ones in the literature and for other cases they also improve upon the best known ones. All instances and solutions are made available on the web for future comparisons.

1 Introduction

We consider a real-world container loading problem, arising from an industrial application, that includes several practical features, such as multiple containers, box rotation, and bearable weight. In addition, the problem takes into account the possibility that the boxes must be delivered in different places (*multi-drop*), thus setting additional constraints on the order of the boxes in the container.

DIEGM, University of Udine
via delle Scienze 208, I-33100, Udine, Italy
E-mail: {sara.ceschia,schaerf}@uniud.it

Our problem belongs to the family of *Container Loading Problems* which counts several known variants depending on the number of containers, the objective function, and the side constraints.

We first classify our problem in the existing literature, reaching the conclusion that it represents a combination of features that has not been explored already. Therefore, no previous approaches are available and, consequently, no benchmark instances.

We solve the problem using a local search approach which works of an indirect search space composed of sequences of boxes, rather than on their physical position. The actual placing of the boxes is performed by a heuristic procedure that loads the containers according to the constraints, exploiting as much as possible the presence of sub-sequences of boxes of the same type.

We test our solver on a set of real-world instances provided by our industrial partner BeanTech s.r.l. (`http://www.beantech.it`). The outcome is that our solution techniques have been able to find very good solutions on a large variety of practical cases, which improve significantly upon the previous heuristic solution developed by BeanTech.

All instances and solutions are available from our dedicated web site `http://satt.diegm.uniud.it/3DPacking`. The web site contains also an application for validating and visualizing new solutions, so as to allow everybody to perform a fair comparison with ours.

In order to have a more measurable assessment of the quality of our solver, we test it also on available benchmarks from the literature. The benchmarks refer to much simpler problems with respect to the one we address, and therefore we solve them by adapting our software, mainly discarding some of the features.

The outcome on the benchmarks is that the for the single-container case the results are in general comparable with the ones in the literature, although, they are still worse than the best known ones. On the contrary, for the multi-container case, we outperform all previous approaches obtaining new best known results.

2 Problem Description

We introduce the problem in two stages: we first describe (in Section 2.1) what we call the *basic problem*, which does not consider the multi-drop feature. In fact, the latter is the most complex feature to be dealt with and it modifies the model significantly; consequently, it is presented and modeled separately (in Section 2.2).

2.1 Basic Problem

The main entities involved in the problem are *container types* and *box types*:

Container types: each container type is characterized by: dimensions, number of containers, weight limit, and fixed cost of use.
Box types: each box type is characterized by: dimensions, allowed rotations, number of boxes, weight, cost, and bearable weight for each face.

The specific features that are involved in the different formulations have been classified by Bischoff and Ratcliff (1995). According to their terminology, our problem comprises the following ones:

Box rotations (BR): the boxes may be rotated in orthogonal directions; possible rotations are stated for each box type.
Load bearing strength (LBS): the maximum weight per unit area which a box can uphold depends on its type and its vertical orientation.
Full Support (FS): a box should not be placed on top of another with a smaller base area; more specifically, both dimensions of the base of the box above must be smaller or equal of the ones of the box below.
Container weight limit (CWL): the sum of the load must not exceed the weight limit of the container.

Using the classification system proposed by Wäscher et al. (2007), our problem could be described as a *Three-Dimensional Regular Multiple Heterogenous Knapsack Problem/ Three-Dimensional Regular Multiple Bin Packing Problem*, that means that:

- we deal with three-dimensional items (*Three-Dimensional*),
- at times we use all the containers available to load the maximum volume of boxes so that some boxes are left outside (*output maximization*), at others we use only a subset of containers to load all the boxes (*input minimization*),
- we have many boxes of many different dimensions (*strongly heterogeneous assortment of small items*), and
- multiple containers of different dimensions (*weakly heterogeneous assortment of several large objects*),
- boxes have a rectangular shape (*regular shape of small items*)

The objective function f is the combination of the following components:

C1. **Boxes not loaded:** cost of boxes that do not fit in the containers.
C2. **Container cost:** fixed cost for using each container.
C3. **Empty linear space:** linear space, in the depth direction, that is empty (available for loading unforeseen items).

More specifically, we define f as follows:

$$f = w_1 f_{C1} + w_2 f_{C2} + w_3 f_{C3}$$

where f_{Ci} is the actual cost of the component Ci, and w_i is the corresponding weight.

In order to simplify the complex process of setting the weights w_i and, at the same time, to have an immediate grasp of the costs, we decide to

represent the costs directly in a real currency, Euro (€) in our case. To the aim of evaluating the monetary costs of each component, we had to interview our industrial partner and the domain experts. Some costs are relatively easy to be established, such as the cost of renting a truck; on the contrary, some others are rather intangible and we could estimate only the approximate cost. For example, the missed delivery of a box leads to the dissatisfaction of the client thus involving the risk of loosing the client, whose cost is difficult to be interpreted in monetary terms.

It is worth noticing, that the objective function is almost *hierarchical*. In fact, intuitively it is more important to deliver the boxes (C1) than to save a container (C2). In turn, saving a container is more important than having a large empty linear space in all containers (C3). However, the function cannot be treated as totally hierarchical because there are situations in which costs related to C2 are more prominent than those of C1. For example, we prefer to leave a few boxes undelivered rather than moving one container only for them.

To design an objective function that is fine-grained enough, but can be represented in the integer domain (so as to use the faster integer arithmetic), we set the unit of cost to the *thousandth of €*.

2.2 Complete Problem

In the complete problem, boxes loaded in the container may be delivered to different destinations. The input data are extended by including for the box types also the identifier of the destination $d \in D$, where D is the set of destinations. In the case of boxes of the same dimensions, allowed rotations, number of boxes, weight, cost, and bearable weighs, but different destinations d_1 and d_2, we create separate box types having all data (but the destination and the quantity) identical.

There are two implications of this extension. First, we now aim also to load the containers in such a way that they make the minimum number of stops; this is obtained by preferring solutions that group boxes to the same destination in the same container(s). Second, containers with boxes with multiple destinations must be loaded in such a way that boxes belonging to the earlier destinations can be unloaded without having to move the other ones.

Regarding the first issue, it is clear that, besides the other constraints and objectives, there is also an underlying vehicle routing problem (VRP) (see, e.g., Toth and Vigo 2002) to be solved. However, for the practical situations of our industrial partner, distances are very limited and thus the traveling costs are not taken into account. The additional objective that we include is then to minimize the total number of stops of the containers. Ideally, each container should go to one single destination, whereas destinations can be served by many containers.

We therefore introduce a new cost component:

C4. **Container stops:** sum of the number of distinct destinations of the boxes for each container.

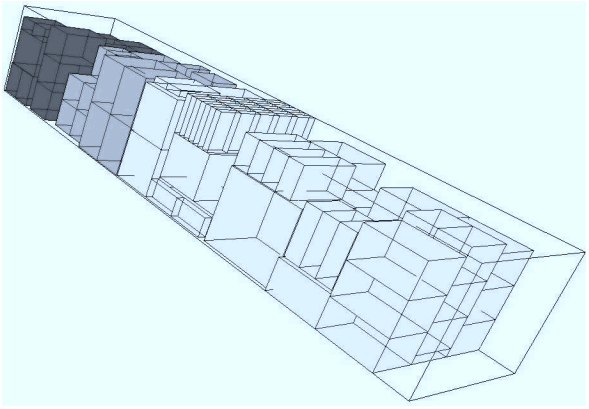


Fig. 1 Multi-drop container loading.

Regarding the second issue, we model it by imposing the constraint that all boxes of a destination should be able to be unloaded *without moving any* of the ones of the subsequent destination.

Multi-drop (MD): if a container carries consignments of different destinations, the boxes should be loaded in such a way that it is possible to set an order of delivery such that no box to a later destination needs to be moved to unload the boxes of the earlier ones.

An example of a container loaded under the multi-drop constraint is shown in Fig. 1, where the three destinations are highlighted by the different gray levels.

As will be explained in Section 4, this is obtained by allowing only loading sequences that have boxes strictly ordered by destination. Since the loading strategy places the sequences in layers starting from the bottom of the container towards the door, this ensures that Constraint MD is always satisfied.

3 Related Work

Most of the work of the recent years has been focused on the development of techniques to solve the container loading problem, where the objective is to achieve the highest volume utilization for a single container.

Heuristics have been proposed by Bischoff et al. (1995) and Pisinger (1998, 2002): the first ones fill the container in horizontal layers from the floor upwards, the others in verticals walls from the rear to the door. Eley (2002) and Fanslau and Bortfeldt (2010) use tree search algorithms that load boxes arranged in cuboid blocks. The second ones have recently published the best known results on available instances for both cases with and without the full support constraint.

Regarding metaheuristic techniques, Bortfeldt and Gehring (2001) and Gehring and Bortfeldt (2002) use genetic algorithms, whereas Bortfeldt et al.

(2003) and Mack et al. (2004) propose respectively Tabu Search and Simulated Annealing methods. GRASP approaches have been presented by Moura and Oliveira (2005) and Parreño et al. (2008). Finally, Parreño et al. (2010) have applied a Variable Neighborhood Search algorithm.

Bischoff and Ratcliff (1995) have described a variety of additional issues that may be important in many real world applications, for which considering the volume utilization as the sole objective is inappropriate and they have proposed a heuristic approach previously used to loading pallets in Bischoff et al. (1995).
Several authors (e.g., Davies and Bischoff 1999; Bortfeldt and Gehring 2001; Eley 2002) have considered the additional issue of balancing the weight inside the container. Ratcliff and Bischoff (1998), Davies (2000) and Bischoff (2006) have studied the case that boxes have different load bearing strengths so that each box can support a different weight depending on its orientation.

The *multiple container loading problem* (MCLP) was firstly introduced by Ivancic et al. (1989). The authors solved it using a sequential loading approach and they published 47 benchmark instances with all the containers of the same size. Bischoff et al. (1995) modified their heuristic for a single container, applying it in the case of multiple containers. Bortfeldt (2000) proposed a Tabu Search approach to cater for the multi-container case and Eley (2003) a bottleneck assignment approach.

Raidl and Kodydek (1998) introduced the *multiple container packing problem* (MCPP) where a subset of items, each with a value associated, should be packed in containers such that the total value of the selected items is maximized. In Raidl and Kodydek (1998) and Raidl (1999), they present a genetic algorithm approach, whereas Sang-Moon et al. (2008) propose an evolutionary algorithm.

The *three-dimensional loading capacitated vehicle routing problem* (3L-CVRP) is a combination of vehicle routing and three-dimensional loading. The problem was introduced by Gendreau et al. (2006) and calls for the determination of a set of routes traveled by a fleet of identical vehicles for delivering items to customers, minimizing the total travel cost. Items are rectangular boxes that can be rotated in any orthogonal direction; they are split into fragile and non-fragile items, such that no non-fragile can be placed on the top of a fragile one. In addition all items required by a customer must be placed on the same vehicle. Gendreau et al. (2006) propose a Tabu Search scheme that explores the neighborhood by moving a client from one route to another and then uses an inner greedy packing heuristic to find a feasible solution that minimizes the used length. Tarantilis et al. (2009) present an algorithm which combines Tabu Search and Guided Local Search to build the routes, whereas the loading characteristics are tackled by a bundle of six different simple packing heuristics. They compare their solver on benchmark instances published by Gendreau et al. (2006) improving previous best solutions. Fuellerer et al. (2010) also improve the Tabu Search approach of Gendreau et al. (2006) using an ACO algorithm with a very simple but fast inner packing heuristic. Finally, Wang et al. (2010) have published the state of the art results using Tabu Search

in conjunction with the Deepest-Bottom-Left-Fill and the Maximum Touching Area packing heuristics.

Recently Moura and Oliveira (2009) have introduced the *vehicle routing with time windows and loading problem* (VRTWLP) and they have published a set of instances for benchmarking. In this new problem the vehicle loading order is the inverse of the clients visit order (as in LIFO strategy) and the demand of each client must be packed together in such a way that the total loading volume must not exceed the container limits. A demand is composed by a set of different box types, each one characterized by its physical dimensions, weight and orientation constraint. They present two different solution techniques: the sequential method which maps out the vehicle routes and the container loading at the same time, and the hierarchical method where the loading is considered as a sub-problem of the VRP. For this problem Bortfeldt and Homberger (2008) have proposed a two-stage heuristic "packing first, routing second", achieving high quality results.

4 Solution for the Single Destination Problem

We first introduce the solution technique for the problem with a single destination, as described in Section 2.1. The multi-drop case is dealt with in Section 5.

Our solution technique is not a "pure" local search approach in the sense that local search works on the space of sequences of boxes to be loaded. The actual load is obtained by means of a specialized procedure, called *loader*, which is invoked at each iteration for all containers involved in the move. The loader inserts the boxes in the container using a deterministic heuristic strategy which produces a load that is feasible according to all our constraints.

4.1 Preprocessing

In some instances, the number of available containers is largely in excess with respect to the boxes to be loaded. In these cases, the inclusion of all containers in the search space would only result in a waste of time for the solver.

The task of the preprocessor is to reduce the set of available containers to a set that any reasonable solution could use. The preprocessor works only on the basis of the volumes of the containers and the total volume of the boxes, without taking into account the actual boxes.

It makes use of a parameter, called α, which is an estimation of the maximum ratio between the volume of a container and the volume of the boxes it actually carries. The value of α is based on runs on known instances, however it can be adjusted in subsequent solutions of a single instance.

The preprocessor uses a simple greedy algorithm, which works as follows.

1. Let V_t be the total volume of the boxes. Set the volume V to be loaded to $V = \alpha V_t$.

2. Select the container with the lowest specific cost defined as the ratio of the fixed cost of use to the volume of the container; set $V = V - V_c$ where V_c is the volume of the container.
3. If $V < 0$ exit, otherwise go back to Step 2.

Containers that are not selected are removed from the input data of the solver.

4.2 Search Space

We use a local search procedure in which a state in the search space is composed by a set of sequences, one for each container. Each element of a sequence is a *block*, which is a triple composed by box type, rotation and number of boxes. A block is also assigned an integer-valued identifier (which is unique in the state).

In any state of the search space all input boxes must be included in exactly one block. However, it is possible that the loader leaves some boxes *in blocks at the end of a sequence* outside the container. This can happen either because there is no physical room for the box or because the weight limit of the container is exceeded. We call the set of unloaded boxes the *tail* of the container, and the sum of the costs of all tails constitute the component C1 of the objective function.

4.3 Loader

The loader is based on the wall building approach by George and Robinson (1980) mentioned in Section 3. It fills the container in a number of *layers* across the depth of it. Every layer is divided in vertical *stacks* and each stack is then packed by consecutively inserting boxes, keeping fixed the loading order of the blocks.

The depth of a layer is set to be the depth of the first box that is inserted in that layer. For all the following boxes, if they do not fit in that depth they are moved to a new layer.

Similarly, if a box does not fit on top of the previous one in a stack, it is moved to a new stack or (if there is not enough room) to a new layer. A box might not fit in a stack either because its height exceeds the container ceiling, or because the base area of the box below is insufficient in at least one dimension, or because the weight is not bearable by one of the boxes below in the stack.

As already proposed by George and Robinson (1980) and Moura and Oliveira (2005), the layers are *flexible* in the sense that boxes are allowed to slide down in the previous layer if there is enough room. Fig. 2 shows the loading of a container viewed from the top, and it helps to clarify this point. The sliding down of the boxes of layer 2 has created the space for the two boxes marked with the cross, which are inserted in layer 2 as well.

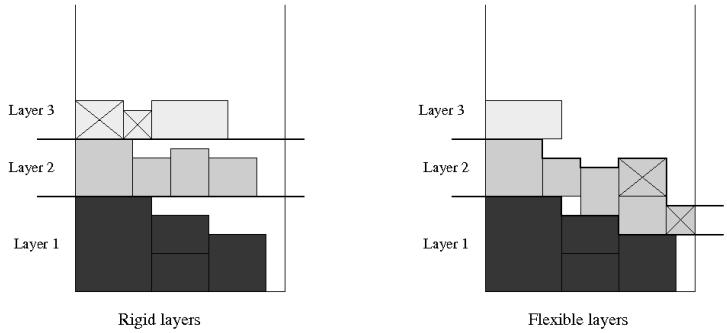


Fig. 2 Loader with rigid and flexible layers.

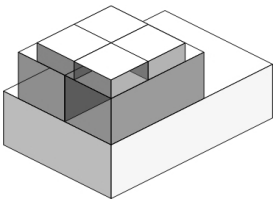


Fig. 3 Stack construction with aside boxes.

Only for boxes belonging to the same block (same dimensions and rotation), we try to put a box *aside* the previous one if the composite base area is inside the one of the box below and the weight is bearable. See in Fig. 3 an example in which two boxes are side by side above a single one, and four boxes are in turn above the merge of these two.

We do not put aside boxes of different types because of stability problems; that is, the boxes above might not have an horizontal base to lay on. Note that boxes of different types could in principle be placed aside if they are on the highest layout (just below the ceiling of the container). However, experiments with this option showed that this does not lead to better solutions.

Notice that the loader could be improved by selecting the next box to be placed in the container using some sort of *best fit* strategy, instead of using the strict order induced by the sequence. The loader however is not the "optimizer", but it is just a module of the overall procedure. Indeed, our idea is precisely that the loader should be simple and fast, and the discovery of improvements is totally demanded to the main local search procedure that performs the changes in the sequence given to the loader.

4.4 Cost Function

All constraints are enforced by construction, thus the cost function that guides the search is the same as the objective function of the problem; it is a weighted combination of C1, C2, and C3 (remember that C4 is related only to the multi-drop case).

However, an auxiliary component C5 is added to it in order to lead the search toward states that use less containers. Specifically, the component C5 sums up for each container the *square* of its volume minus the *square* of the volume of the boxes loaded. We use a quadratic term in order to privilege solutions with asymmetric loading of containers, so that it would eventually lead to freeing completely the least loaded ones, rather than keeping the load balanced among them.

It is worth noticing that components C2 and C3 are not sufficient to obtain this behavior. In fact, they come into play only on the removal of the last block from a container and the last block of the layer, respectively. In the other states, they do not "push" in the direction to have a container with less blocks. Notice also that some containers are already removed by the preprocessor; however, the preprocessor works on a rough overestimation of the loading level (α), whereas here we work on the actual situation.

Indeed, experiments without C5 show a substantial increase of the number of containers used in the final solutions, and a corresponding increase of the total cost.

4.5 Neighborhood Relation

The neighborhood consists in moving a block, or a portion of it, in a different container and/or different position, possibly with a different orientation. Fig. 4 shows an example of a move in which part of a block is moved from Container 1 to Container 2.

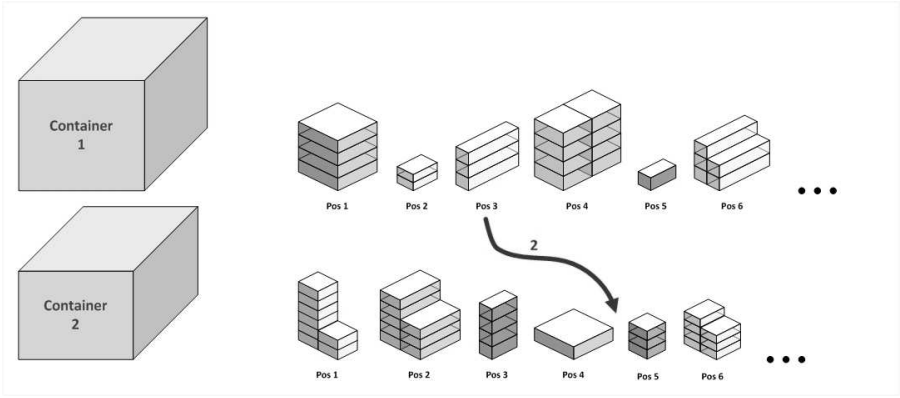
A move is thus represented by five attributes: block identifier, new container, new position, new rotation, and quantity of boxes moved. The quantity varies from 1 to the size of the block.

When only a portion of a block is moved (i.e., the quantity is less than the size of the block), the block is split and thus a new block with a new identifier is created. Conversely, when a block is moved and the two blocks before and after it are compatible (i.e., same box type and rotation), they are merged summing up their quantities and keeping the identifier of the block before. Similarly if a block is moved to a position adjacent to a compatible one, they are merged, keeping the identifier of the one already there.

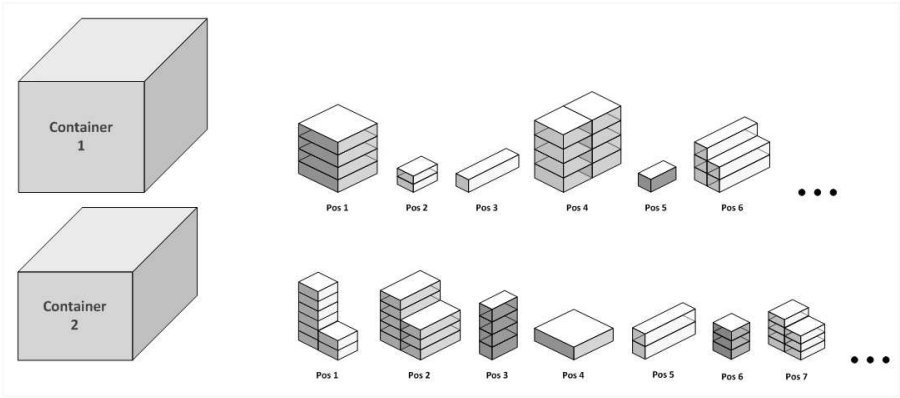
The neighborhood is *restricted* in such a way to remove moves that are clearly useless for the local search. In detail, we exclude the following types of moves:

Tail-to-tail moves: a move that transfers a block from the tail of a container to the tail of another one. This kind of moves do not change the cost of the objective function.

Duplicated moves: a move that leads to the same state of another move. This are the moves that move a full block after the one that immediately follows it: the same movement is already obtained by moving the second block one position backward (Figure 5).



Initial state



Final state

Fig. 4 A example of move.

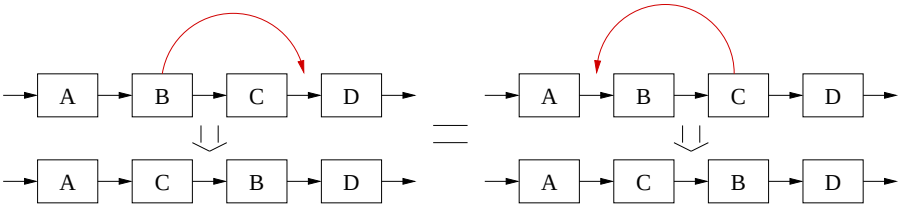


Fig. 5 An example of duplicated move

Null moves: a move that leads to the same state as the current one. These are the moves that either the new container, the new position, and the new rotation are all the same of the current ones of the block or they move part of a block just after the block itself (which then is rebuild by merging).

When a move is evaluated, the loader is called only on the two containers (or one if the move is internal to one container) involved in the move. In addition, only the part of the load starting from the stack that includes the

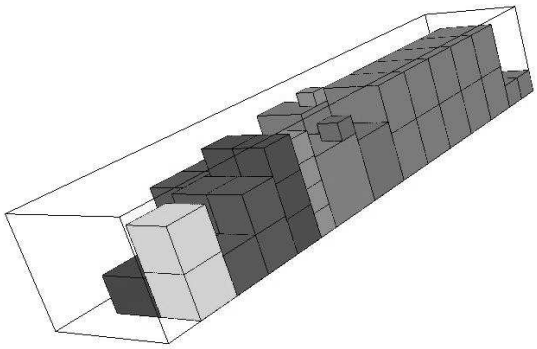


Fig. 6 Example of initial solution.

box being moved is recomputed. The preceding part is not affected by the move and thus it is left unchanged.

4.6 Initial Solution

The initial solution is obtained by creating one single block for each box type. The container, the rotation, and the position in the container of each block is selected at random. The rotation is selected only among the feasible ones for that box type.

The rationale behind this choice of having one block per type is that intuitively we should keep the blocks as big as possible, because it is easier to pack boxes of the same type when they are close to each other. The local search procedure would then split the blocks, when this leads to better solutions. Figure 6 shows an example of initial solution.

4.7 Metaheuristics

We designed two metaheuristic strategies: Simulated Annealing (Kirkpatrick et al. 1983; Eglese 1990; Aarts et al. 1997) and Tabu Search (Glover and Laguna 1997). The specific version used is described in the following:

Simulated Annealing (SA): At each iteration a random neighbor is selected. The move is performed either if it is an improving one or according to an exponential time-decreasing probability. In detail, if the cost of the move is $\Delta f > 0$, the move is accepted with probability $e^{-\Delta f/T}$, where T is a time-decreasing parameter called *temperature*. At each temperature level a number σ_N of neighbors of the current solution is sampled and the new solution is accepted according the above mentioned probability distribution. The value of T is modified using a *geometric* schedule, i.e., $T' = \tau \cdot T$, in which the parameter $\tau < 1$ is called the *cooling rate*. The search starts at temperature T_0 and stops when it reaches T_{min}.

Tabu Search (TS): At each step a subset of the neighborhood is explored and the neighbor that gives the minimum cost value becomes the new solution independently of the fact that its cost value is better or worse than the current one. The subset is induced by the *tabu list*, i.e., a list of the moves recently performed, whose *inverse* moves are currently forbidden and thus excluded from the exploration. The tabu status can be overridden by the (standard) *aspiration criterion*: a tabu move that leads to a solution better than the current best one is accepted. Our tabu search implementation employs a dynamic short-term tabu list, called Robust Tabu Search in (Hoos and Stützle 2005), so that a move is kept in the tabu list for a random number of iterations in the range $[k_{min}, \ldots, k_{max}]$. The search stops after a fixed number of *idle* iterations (ii_{max}), i.e. iterations without an improvement (stagnation detection).

5 Solution Technique for the Multi-Drop Problem

For the multi-drop case, the idea is to solve a multi-drop instance by solving a sequence of *sub-instances* obtained by adding one destination at the time.

Letting $D = \{d_1, \ldots, d_n\}$ be the set of destinations, we sequentially solve n sub-instances, such that sub-instance i (with $i = 1, \ldots, n$) is obtained considering only the box types belonging to the destinations d_j, such that $j \leq i$.

Each sub-instance i is solved by the technique presented in Section 4, with some modifications to take care of the MD constraint and the cost component C4. The modifications are the following:

Reserved Containers: If in the final solution of sub-instance i a given container c is fully loaded, then c is *reserved* to the destinations that are in c in that solution. This means that during the solution of all sub-instances j, with $j > i$, only box types of those destinations can be inserted in c.

Incremental Initial Solution: The initial solution of sub-instance i is obtained by starting from the best solution of instance $i - 1$ and adding to it the blocks of destination d_i at random only at the end of the sequences of non-reserved containers. For sub-instance 1, the initial solution is obtained in the same way of Section 4.6.

Neighborhood Restrictions: The local search procedure takes into account the multi-drop constraint by prohibiting to move a box of destination d_i in container c in position p if one of the following conditions holds:

- the container c is reserved and d_i does not belong to its reservation list,
- a box of destination d_j (with $j < i$) is in container c in a position which follows p.

This technique proved experimentally to be more performing than solving the overall problem with a single local search step.

Family		Conditions		Size				Cost
Name	#I			#CT	#C	#BT	#B	components
CS1	71	$f_{c1} = 0$	$f_{c2} = \gamma$	1–3	1–100	1–124	4–2760	C3
CS2	12		$f_{c2} < \gamma$	1–2	100–200	10–90	23–212	C2 and C3
CS3	31	$f_{c1} > 0$	$f_{c2} = \gamma$	1-3	1–100	5–97	27–1612	C1
CS4	3		$f_{c2} < \gamma$	1-2	2–5	2–47	82-1439	C1 and C2

Table 1 Families of instances.

6 Experimental Analysis

In this section, we first describe the instances used in the experiments. Secondly, we show the results of our methods on these instances. Lastly, we show the comparison between our methods and the ones in the literature on simpler problems.

6.1 Instances

We experiment on a set of 117 real-world instances coming from the clients of our industrial partner BeanTech. All instances are available at the URL `http://satt.diegm.uniud.it/3DPacking`.

The instances exhibit different container types and a high variability in terms of number of box types and quantity of each type. This large variability can be expressed in terms of the ratio between the total volume of the boxes and the total volume of the containers, that we call β. Normally if $\beta \ll 1$, we expect that all boxes fit into the containers, and consequently, $f_{C1} = 0$ and the other components become significant. Conversely, when $\beta \geq 1$, the cost function is dominated by f_{C1} and the other components are almost negligible. For the values $\beta < 1$ (but $\beta \not\ll 1$) it depends on the specific instance whether f_{C1} is zero or not.

A further distinction can be made between instances in which all containers are always used, for which f_{C2} is fixed to the total cost of the containers (called γ) and the ones for which f_{C2} varies from run to run, and is thus meaningful.

We thus classify our instances in four families, called CS1–CS4, based on the two conditions that the best known solution has $f_{C1} = 0$ or not, and $f_{C2} = \gamma$ or not, respectively. Table 1 shows for each family the number of instances belonging to it, the ranges in terms of containers (#C), container types (#CT), box types (#BT), and total boxes (#B), and the cost components that are most meaningful for that family.

6.2 Parameter Setting

We now discuss the settings of the parameters of the metaheuristics. A tuning phase has been performed to obtain the configuration of values that would be the best for the entire data set CS1–CS4. The settings that obtained the best results are the following:

	C3			
Technique	Avg	Dev	t-test	Running times (secs)
SA	**338.256**	**9.177**	–	**39.41**
TS	351.999	10.235	0.044	83.05

Table 2 Results of the C3 component on family CS1.

	C2		C3			
Technique	Avg	Dev	Avg	Dev	t-test	Running times (secs)
SA	**0.088**	**0.0082**	**158.802**	**4.421**	–	**38.95**
TS	0.096	0.016	161.263	4.557	0.16	86.42

Table 3 Results of the C2 and C3 components on family CS2.

	C1			
Technique	Avg	Dev	t-test	Running times (secs)
SA	**2572.517**	**167.133**	–	**52.78**
TS	2662.959	164.012	0.060	100.31

Table 4 Results of the C1 component on family CS3.

for SA: $T_0 = 50$, $T_{min} = 0.001$, $\sigma_N = 1000$, and $\tau = 0.998$
for TS: $k_{min} = \lfloor \delta \rfloor / 5$, $k_{max} = k_{min} + 2$, and $ii_{max} = 500$, where δ is the number of blocks in the current solution; as for the inverse relation, a move is prohibited if it involves the same block of a move in the tabu list.

Notice that δ varies from state to state, so that the tabu list length is *adaptive*. The results of the following section are obtained with the above configurations.

6.3 Experimental Results

We present in Tables 2–5 the results separately for each family. As mentioned in Section 2.1, we chose to express the cost of the components of the objective function in €.

To compare the results we use the *Student's t-test*, provided that the underlying distributions can be assumed to be normal and independent (Venables and Ripley 2002). If the calculated p-value is below the threshold chosen for statistical significance (typically $p < 0.05$), then the *null hypothesis*, which states that the two groups do not differ, is rejected in favor of an alternative hypothesis, which states that an algorithm is superior to another on the proposed instances.

The column t-test shows the p-value of the comparison between best configuration (marked with a dash) and the other one.

Looking at Tables 2–4, it is clear that for these instances SA is the technique that works better than TS, although the confidence is not always high enough to reach a definitive conclusion. On the other hand, TS preforms well only on instances of family CS4 which is made of only 4 instances.

	C1		C2			
Technique	Avg	Dev	Avg	Dev	t-test	Running times (secs)
SA	715.346	119.590	0.183	0	0.277	97.39
TS	**695.717**	**156.689**	**0.183**	**0**	–	**153.37**

Table 5 Results of the C1 and C2 components on family CS4.

Family		Size			Features					Objectives			
Name	#I	#C	#BT	#B	BR	LBS	FS	MD	CWL	C1	C2	C3	C4
IMM	47	2–55	2–5	47-150	√	—	√	—	—	—	√	—	—
BR	1500	1	3–100	69–476	√	—	√	—	—	√	—	—	—

Table 6 Features of the families IMM and BR1–BR15.

The reason for these bad performances of TS is in our opinion mainly due to the computational cost of the full exploration of the neighborhood, which evidently turned out to be less effective than the random selection of SA.

The low value of the component C2 in Table 3 that computes the fixed cost of using the containers, can be explained by the fact that, for the instances of family CS2 there are always available *in-house* containers (owned by the company), whose cost was set to the fictitious value of 1 thousandth of € by the operator of the company.

6.4 Comparison with Related Work

Given that the problem is new, it is not possible to compare with other researchers on this problem. However, in order to have an assessment of the quality of the solver, we use it to solve simpler problems for which previous results are available.

To this aim we use two public benchmarks: the family IMM proposed by Ivancic et al. (1989), and BR defined by Bischoff and Ratcliff (1995).

Table 6 summarizes for each family the number of instances (#I), the ranges of the instances in terms of containers (#C), box types (#BT), and total boxes (#B). The following columns represent the problem features, and the symbol √ means that it is present in the family. Finally, we have the cost components, and here √ means that it has been considered by the papers that solve the instances of that family. It is evident that the families IMM and BR cover only a small subset of the features of this work set and exhibit a much smaller variability of size.

We start discussing the results of IMM instances, which consider one container type (i.e., all containers are identical), and as objective the number of containers used (i.e., C2 with all costs equal to 1).

We report in Table 7 the number of containers used in the best solution found by our SA solver, along with the results in the literature. The average running time for a trial of a single instance for SA is 289.37 secs. The best known result is shown in bold; in addition, when it is proven optimal (by Eley

Test case	Ivancic et al. (1989)	Bischoff and Ratcliff (1995)	Bortfeldt (2000)	Eley (2002)	Eley (2003)	SA (us)
IMM1	26	27	**25***	26	**25***	**25***
IMM2	11	11	**10**	**10**	**10**	**10**
IMM3	20	21	20	22	20	**19**
IMM4	27	29	28	30	**26***	**26***
IMM5	65	61	**51**	**51**	**51**	**51**
IMM6	**10***	**10***	**10***	**10***	**10***	**10***
IMM7	**16***	**16***	**16***	**16***	**16***	**16***
IMM8	5	**4***	**4***	**4***	**4***	**4***
IMM9	**19***	**19***	**19***	**19***	**19***	**19***
IMM10	**55***	**55***	**55***	**55***	**55***	**55***
IMM11	18	19	18	18	**17**	**17**
IMM12	55	55	**53***	**53***	**53***	**53***
IMM13	27	**25**	**25**	**25**	**25**	**25**
IMM14	28	**27***	28	**27***	**27***	**27***
IMM15	**11***	**11***	**11***	12	**11***	**11***
IMM16	34	28	**26***	**26***	**26***	**26***
IMM17	8	8	**7***	**7***	**7***	**7***
IMM18	3	3	**2***	**2*♭**	**2***	**2***
IMM19	**3***	**3***	**3***	**3*♭**	**3***	**3***
IMM20	**5***	**5***	**5***	**5*♭**	**5***	**5***
IMM21	24	24	21	26	**20**	**20**
IMM22	10	11	9	9	**8***	**8***
IMM23	21	22	**20**	21	**20**	**20**
IMM24	**6**	**6**	**6**	**6**	**6**	**6**
IMM25	6	**5**	**5**	**5**	**5**	**5**
IMM26	**3***	**3***	**3***	**3***	**3***	**3***
IMM27	**5**	**5**	**5**	**5**	**5**	**5**
IMM28	**10**	11	**10**	**10**	**10**	**10**
IMM29	18	**17**	**17**	18	**17**	**17**
IMM30	24	24	**22**	23	**22**	**22**
IMM31	**13**	**13**	**13**	14	**13**	**13**
IMM32	5	**4***	**4***	**4***	**4***	**4***
IMM33	5	5	5	5	5	**4**
IMM34	9	9	**8**	9	**8**	**8**
IMM35	3	3	**2***	**2***	**2***	**2***
IMM36	18	19	**14***	**14***	**14***	**14***
IMM37	26	27	**23***	**23***	**23***	**23***
IMM38	50	56	**45**	**45**	**45**	**45**
IMM39	16	16	**15**	**15**	**15**	**15**
IMM40	9	10	9	9	**8**	**8**
IMM41	16	16	**15**	**15**	**15**	**15**
IMM42	**4***	5	**4***	**4***	**4***	**4***
IMM43	**3***	**3***	**3***	**3***	**3***	**3***
IMM44	4	4	**3**	4	4	**3**
IMM45	**3**	**3**	**3**	**3**	**3**	**3**
IMM46	**2***	**2***	**2***	**2***	**2***	**2***
IMM47	4	**3***	**3***	**3***	**3***	**3***
All cases	763	763	705	721	699	**696**

Table 7 Results for family IMM.

2003) it is marked with *. The values marked with the symbol ♭ have been inferred by us, as in (Eley 2002) it is reported a value that is lower than the lower bound subsequently proven by Eley (2003) himself. It is evident from Table 7 that our solver has been able to obtained at least the best known result for all instances.

Keeping in mind that we can obtain only an approximate comparison because we do not have information about the running times and the distributions of all the other solvers, looking at the bottom line of Table 7, it is clear

that our results improve significantly on the original works by Ivancic et al. (1989) and Bischoff and Ratcliff (1995). We also improve upon the more recent works by Bortfeldt (2000) and Eley (2003). With respect to Eley (2003) we are ahead in three instances (IMM3, IMM33, and IMM44), and for two instances (IMM3 and IMM33), we have obtained the new best known solution.

All our best solutions to the IMM instances, along with the input files in our format, are available at our web site `http://satt.diegm.uniud.it/3DPacking` for verification.

The BR dataset is divided into 15 families of 100 instances each, depending on the number of box types (shown in parentheses in Table 8). All instances have one single container and one single objective, namely C1. No cost is associated to the boxes, therefore the objective function is based on the box volume (volume utilization).

The results are shown in Table 8. Again, the comparison is approximate because we do not have access to all data about the distributions and the running times of all the others. In any case, the outcome is that our solver improves on the original results of Ratcliff and Bischoff (1998) and achieves results quite close to the heuristic of Eley (2002) and the GRASP approach of Moura and Oliveira (2005). Unfortunately though, it is clearly outperformed by Bortfeldt and Gehring (1998), Bischoff (2006) and the recent results by Fanslau and Bortfeldt (2010).

However, all the techniques reported in Table 8 are specifically designed and tuned for the case that involves only one container, whereas our technique solves the more general problem that deals with bearing weights, different container types, multi-drop and other features, as previously described in Section 2.2. In addition, our solution is designed to work equally well in the case of weakly heterogeneous cargoes, with large quantities of identical boxes, and strongly heterogeneous ones composed of many different box types.

For instance, Eley (2002) takes advantage of the fact that the loading is weakly heterogeneous by considering a limited number of potential arrangement of pre-designed homogeneous blocks of boxes. Fanslau and Bortfeldt extend the classical block build approach in order to face also with a situation of strongly heterogeneous cargo. At each stage of the solution process their technique generates the packing patterns that come out from the arrangement of blocks made up of different box type and orientation. Nevertheless, this procedure could result impracticable in case of simultaneous loading of multiple containers, thus the number of possible arrangements for each residual space would explode.

It is clear that these approaches would not be applicable in our case, because our instances have a much larger variability in terms of number and types of boxes and containers, as clearly shown in Table 1.

Other authors (e.g., Mack et al. 2004) have solved these instances finding also higher percentages of volume occupation. For example, Parreño et al. (2010) reach an average filling of 94.53 for cases BR1-7. However, they consider a different notion of stability, in which the base of a box does not need to be

Test case (# box types)	Bischoff and Ratcliff (1995)	Gehring and Bortfeldt (2002)	Eley (2002)	Moura and Oliveira (2005)	Bischoff (2006)	Fanslau and Bortfeldt (2010)	SA (us)
BR1(3)	83.79	88.1	88.05	89.07	89.39	**94.51**	90.31
BR2(5)	84.44	89.56	88.44	90.43	90.26	**94.73**	90.75
BR3(8)	83.94	90.77	89.23	90.86	91.08	**94.74**	90.52
BR4(10)	83.71	91.03	89.24	90.42	90.9	**94.41**	89.84
BR5(12)	83.8	91.23	88.99	89.57	91.05	**94.13**	89.16
BR6(15)	82.44	91.28	88.91	89.71	90.7	**93.85**	88.73
BR7(20)	82.01	91.04	88.36	88.05	90.44	**93.2**	87.68
Cases BR1–7	83.45	90.43	88.75	89.73	90.55	**94.22**	89.57
BR8(30)		90.26		86.13		**92.26**	86.36
BR9(40)		89.5		85.08		**91.48**	85.51
BR10(50)		88.73		84.21		**90.86**	84.49
BR11(60)		87.87		83.98		**90.11**	83.58
BR12(70)		87.18		83.64		**89.51**	83.21
BR13(80)		86.7		83.54		**88.98**	82.41
BR14(90)		85.81		83.25		**88.26**	81.91
BR15(100)		85.48		83.21		**87.57**	80.77
Cases BR8–15		87.69		84.13		**89.88**	83.53
All cases		88.97		86.74		**91.91**	86.35

Table 8 Results for family BR.

fully supported by the box below, but its base can be also partly outside it. Same comment can be drawn also for the cutting variant of the algorithm proposed in Fanslau and Bortfeldt (2010), which was able to obtain the current best results with an average volume utilization of 93.8 through all the family BR. For this reason, they are not included in Table 8.

7 Discussion, Conclusions, and Future Work

We have developed a set of techniques for solving a very complex packing problem, including many features and cost components. Our solvers have been able to deal with a broad assortment of different practical situations, ranging from cases in which there is an unlimited availability of containers to cases in which it is not possible to load all the boxes.

With respect to the product currently in list for BeanTech, the solver captures a much richer model as it adds the multi-drop feature, the bearing strength, and the costs of containers and boxes. In addition, it improves or equals its results on all instances. The average improvement is about 13% on the tested instances. For the sake of measurability, all the CS instances are available on the web, along with our solutions and a validator for new solutions.

The experiments have shown that on our complete problem SA outperforms TS in most cases. They also demonstrate that on public benchmarks our results

are very competitive for the multi-container cases, whereas they still worse than some in the literature for the single container ones.

For the future, we plan to undertake the following tasks:

- improve the preprocessor, which currently uses a greedy approach;
- further improve the flexible layers of the wall building approach of the *loader*, in such a way that reduces the free space at the ground level;
- investigate new neighborhoods and (for TS) new adaptive memory strategies;
- experiment with other meta-heuristic techniques.

Acknowledgments

We wish to thank BeanTech s.r.l., and in particular Massimiliano Anziutti, for the support in the modeling of the problem and for providing us the input data. We also thank Stefano Savanelli for developing the packing web site and the 3D container visualizer.

References

Aarts, E. H., Korst, J., and van Laarhoven, P. J. (1997). Simulated annealing. In Aarts, E. H. and Lenstra, J. K., editors, *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester.

Bischoff, E. (2006). Three-dimensional packing of items with limited load bearing strength. *Journal of Operational Research*, 168(3):952–966.

Bischoff, E. E., Janetz, F., and Ratcliff, M. S. W. (1995). Loading pallets with non-identical items. *European Journal of Operational Research*, 84(3):681–692.

Bischoff, E. E. and Ratcliff, M. S. W. (1995). Issues in the development of approaches to container loading. *Omega*, 23(4):377–390.

Bortfeldt, A. (2000). A heuristic for multiple container loading problems. *OR Spectrum*, 22:239–261. In German.

Bortfeldt, A. and Gehring, H. (1998). A tabu search algorithm weakly heterogeneous container loading problem. *OR Spektrum*, 20:237–250. In German.

Bortfeldt, A. and Gehring, H. (2001). A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, 131(1):143–161.

Bortfeldt, A., Gehring, H., and Mack, D. (2003). A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29(5):641–662.

Bortfeldt, A. and Homberger, J. (2008). Packing first, routing second - A heuristic for the vehicle routing and loading problem. In Bortfeldt, A., Homberger, J., Kopfer, H., Pankratz, G., and Strangmeier, R., editors, *Intelligent Decision Support - Current Challenges and Approaches, Intelligente Entscheidungsunterstützung - Aktuelle Herausforderungen und LösungsansätzeIntelligent Decision Support*, pages 91–113. Deutscher Universitäts-Verlag. In German.

Davies, A. (2000). *Approaches to the container loading problem*. PhD thesis, University of Wales, Swansea.

Davies, A. P. and Bischoff, E. E. (1999). Weight distribution considerations in container loading. *European Journal of Operational Research*, 114(3):509–527.

Eglese, R. (1990). Simulated annealing: A tool for operations research. *European Journal of Operational Research*, 46:271–281.

Eley, M. (2002). Solving container loading problems by block arrangement. *European Journal of Operational Research*, 141(2):393–409.

Eley, M. (2003). A bottleneck assignment approach to the multiple container loading problem. *OR Spectrum*, 25(1):45–60.

Fanslau, T. and Bortfeldt, A. (2010). A Tree Search Algorithm for Solving the Container Loading Problem. *INFORMS Journal of Computing*, 22(2):222–235.

Fuellerer, G., Doerner, K. F., Hartl, R. F., and Iori, M. (2010). Metaheuristics for vehicle routing problems with three-dimensional loading constraints. *European Journal of Operational Research*, 201(3):751 – 759.

Gehring, H. and Bortfeldt, A. (2002). A parallel genetic algorithm for solving the container loading problem. *Int. Transactions on Operational Research*, 9(4):497–511.

Gendreau, M., Iori, M., Laporte, G., and Martello, S. (2006). A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40(3):342–350.

George, J. A. and Robinson, D. F. (1980). A heuristic for packing boxes into a container. *Computers and Operations Research*, 7(3):147–156.

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

Hoos, H. H. and Stützle, T. (2005). *Stochastic Local Search — Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA (USA).

Ivancic, N., Mathur, K., and Mohanty, B. (1989). An integer programming based heuristic approach to the three-dimensional packing problem. *Journal of Manufacturing and Operations Management*, 2(4):268–298.

Kirkpatrick, S., Gelatt, Jr, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

Mack, D., Bortfeldt, A., and Gehring, H. (2004). A parallel hybrid local search algorithm for the container loading problem. *Int. Transactions on Operational Research*, 11(5):511–533.

Moura, A. and Oliveira, J. F. (2005). A grasp approach to the container-loading problem. *IEEE Intelligent Systems*, 20:50–57.

Moura, A. and Oliveira, J. F. (2009). An integrated approach to the vehicle routing and container loading problems. *OR Spectrum*, 31(4):775–800.

Parreño, F., Alvarez-Valdes, R., Oliveira, J., and Tamarit, J. (2008). A maximal-space algorithm for the container loading problem. *INFORMS Journal of Computing*, 20(3):412–422.

Parreño, F., Alvarez-Valdes, R., Oliveira, J., and Tamarit, J. (2010). Neighborhood structures for the container loading problem: a VNS implementation. *Journal of Heuristics*, 16(1):1–22.

Pisinger, D. (1998). A tree search heuristic for the container loading problem. *Ricerca Operativa*, 28:31–48.

Pisinger, D. (2002). Heuristics for the container loading problem. *European Journal of Operational Research*, 141:382–392.

Raidl, G. R. (1999). The multiple container packing problem: a genetic algorithm approach with weighted codings. *ACM SIGAPP Applied Computing Review*, 7(2):22–31.

Raidl, G. R. and Kodydek, G. (1998). Genetic algorithms for the multiple container packing problem. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 875–884, London, UK. Springer-Verlag.

Ratcliff, M. and Bischoff, E. (1998). Allowing for weight considerations in container loading. *OR Spectrum*, 20:65–71.

Sang-Moon, S., Lee, S.-W., Yeo, G.-T., and Jeon, M.-G. (2008). An effective evolutionary algorithm for the multiple container packing problem. *Progress in Natural Science*, 18(3):337–344.

Tarantilis, C., Zachariadis, E. E., and Kiranoudis, C. T. (2009). A hybrid metaheuristic algorothm for the integrated vehicle routing and three-dimensional container-loading problem. *IEEE Transactions on intelligent trasportation systems*, 10(2):1524–9050.

Toth, P. and Vigo, D., editors (2002). *The Vehicle Routing Problem*. Monographs on Discrete Mathematics and Applications. S.I.A.M., Philadelpia, PA (USA).

Venables, W. N. and Ripley, B. D. (2002). *Modern applied statistics with S*. Statistics and Computing. Springer, 4^{th} edition.

Wang, L., Guo, S., Chen, S., Zhu, W., and Lim, A. (2010). Two Natural Heuristics for 3D Packing with Practical Loading Constraints. *PRICAI 2010: Trends in Artificial*, pages 256–267.

Wäscher, G., Haußner, H., and Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130.