New Heuristic for the Single Container Loading Problem

Article · January 2017

CITATIONS
0

READS
836

3 authors, including:

Safa Bhar Layeb
University of Tunis El Manar
59 PUBLICATIONS **88** CITATIONS
SEE PROFILE

Amel Jaoua
École Nationale d'Ingénieurs de Tunis
20 PUBLICATIONS **81** CITATIONS
SEE PROFILE

International Journal of Economics & Strategic Management of Business Process (ESMB)
Vol.8, Issue 1, pp. 1-7

New Heuristic for the Single Container Loading Problem

Safa Bhar Layeb[1], Omar Jabloun[2], Amel Jaoua[3]

University of Tunis El Manar, National Engineering School of Tunis
UR-OASIS: Optimization & Analysis of Service and Industrial Systems
BP 37 Le Belvédère, 1002, Tunis, Tunisia

[1]Safa.Layeb@enit.utm.tn
[2]omar.JABLOUN.2@gmail.com
[3]amel.jaoua@polymtl.ca

***Abstract*— Solving the Single Container Loading Problem (SCLP) remains a relevant issue in the transportation and logistics industry. It is faced by the majority of distribution centers and warehouses. The SCLP seeks to pack three-dimensional boxes into a three-dimensional container in order to maximize the total volume utilization.**
For this challenging problem, we propose a new greedy two-step look-ahead procedure by selecting a free space deterministically followed by a block search. We also generalized the proposed heuristic to the single container loading problem with additional constraints to deal with realistic situations.
In order to evaluate its computational performance, the proposed procedure is implemented and tests are carried out on over than 1600 benchmark instances. Our approach performs very well comparing to the most known heuristics from the literature.

***Keywords*— Transportation, Packing, Container loading, Heuristic, Block-building.**

I. INTRODUCTION

Loading rectangular boxes into containers (pallet, Truck, railway …) is a common basic activity in material handling. The usage of containers to pack products has grown hugely in recent years. According to the 2014 annual market review of "Alpha liner" [1], the container ship capacity has grown by a rate equal to 6.3%. Thus, a problem faced by the majority of distribution centers and warehouses is how to load efficiently different items in a single container. Actually, the random and manual packing patterns produce nonstable loading plans leading most of the cases to damages and returns. To address this issue, many recent studies in transportation deal with the Single Container Loading Problem (SCLP).

The SCLP is modeled as an orthogonal packing of rectangular items. The large rectangular parallelepiped is called container and the smaller ones are called boxes or cuboids. It seeks to pack three-dimensional boxes into a three-dimensional container in order to maximize the total volume utilization. Not surprisingly, this challenging problem is known to be an NP-hard problem in the strict sense.

The scope of this work is to propose for this challenging problem a new competitive heuristic method that provides very good solutions in very reasonable computing times. In order to reflect realistic situations, we generalized the proposed framework to fit the weight distribution, stacking, and positioning constraints.

The remainder of this paper is organized as follows. In section 2, we present the project context of this work. In section 3, a literature review is proposed for the SCLP. Then, in section 4, we describe the proposed heuristic procedure. In section 5, we report the results of extensive computational experiments carried out on well-known instances from the literature.

II. PROJECT CONTEXT

Systems, Applications and Products in Data Processing (SAP) [2], is the leader in enterprise resource planning (ERP) in terms of software and software-related services. Although, The SAP and its related products, the Extended Warehouse Management (EWM) [2], optimize during the different stages of the supply chain management (automatic replenishment, yard management…). When it comes to packing in the outbound delivery, there is no algorithm implemented to load efficiently different items in a single container. Unfortunately, random and manual packing patterns produce non stable loading plans leading most of the cases to damages and returns.

In this context, this study is part of a development project aiming to provide a planning procedure when loading single container with mixed products (items with different sizes and dimensions) in the EWM service module of SAP. We are dealing with a problem faced by the majority of distribution centers and warehouses. The objective is to propose and

implement a robust algorithm with an automatic pattern to optimize the loading process.

III. LITERATURE REVIEW

The Single Container Loading Problem with practical constraints is NP-hard in the strict sense since the classical SCLP [3] (orthogonal packing, orientation, no overlap) is reduced to it. Exact algorithms could only solve instances with moderate size ([4]-[5]). Therefore, it is certain to restore to heuristics methods in case of practical situations.

Existing algorithms can be roughly divided into three groups of practical heuristics (not necessarily disjoint). Divide-and-conquer algorithms, which are recursive methods, try to break the container into smaller pieces and use to solve each one recursively before combining them to get a final solution ([6]-[7]). Constructive methods work by repeatedly loading blocks into the container until no further boxes can be loaded. Finally, local search methods start with an existing solution, and then repeatedly apply neighborhood operators to generate newer solutions; an example is illustrated in [8]. Till today the most successful algorithms are the one based on the block-building approach; which forms solutions by repeatedly placing boxes within the container until no box can be packed. A block represents a subset of boxes where the sum of their volume doesn't exceed a maximum threshold (generally set to 98%) of the volume of its bounding cuboid.

The first heuristics was proposed since 1980 [9]. It was a block-building approach that uses vertical layer. In 1995, Bishoff and Ratcliff [10] create a benchmark data set known as the weakly-heterogeneous instances. They suggested selecting high utilization layer and considered the stability. Moreover, some meta-heuristics have been investigated such as the genetic algorithm (e.g. [11]-[12]). In 2003, Bortfeldt et al. [13] developed a parallel Tabu search algorithm; it was a block-building approach. In 2005, Moura and Olivera [14] published a greedy randomized adaptive search procedure based on the work of George and Robinson [9]. Other meta-heuristics for the SCLP were proposed, such as the Simulating Annealing by Jin et al. [15] as well as the variable neighborhood search by Parreño et al. [8].

More recently in 2012, Zhu et al. [16] propose an analytical framework for the block-building approaches. They prove that existing algorithms, from this category, only differ in the decision made for each key. They used a greedy algorithm with a new fitness function that estimates the unused space after a block is loaded. The space in the container is represented same as the work of Parreño et al. [17]. They build an algorithm called the "Maximal Space" (MS). It uses a particular type of simple boxes: the columns and the layers. The blocks are selected based on a parameter δ: the set of boxes that have the top δ % and have the maximal volume or the best fit, are selected.

Zhu and Lim [18] used simple blocks for weakly heterogeneous instances and general blocks for the strongly heterogeneous ones. An indicator h_t determines which kind of blocks to generate. They confirmed that the selection of free space is as important as the selection of a block, and their role is symmetric in the search tree. They compared the Manhattan distance and the corner distance, the first is superior for strongly heterogeneous instances.

In 2014, Araya and Riff [19] offered a constructive approach using a beam search strategy. The proposed algorithm takes key elements from ([16]-[17]) but replaces the overarching strategy. It was an adaptation of the branch-and-bound approach. They handled the full support to guarantee load stability, the bottom sides of each loaded boxes are either fully supported by the container or by the top side of another placed box. Their approach expands the most promising nodes at each level.

Liu et al. [20] present a novel Hybrid-Tabu search approach to the container loading problem. Moreover, their algorithm can solve problems with additional practical constraint such as weight limit and weight distribution when tested over real world data.

Wang et al. [21] consider shipping priority in container loading, where high priority boxes must be loaded before those with low priority. They propose a multi-round partial beam search method that explicitly considers shipping priority when evaluating the potential of partial solutions to solve this problem. Since existing benchmark data for shipping priority covers only weakly heterogeneous instances, they extend the benchmark data to strongly heterogeneous instances.

Lim et al. [22] address the axle weight limit requirements stipulated in the California Vehicle Code related to trucks. A GRASP wall-building algorithm, combined with a linear integer programming models in an overall heuristic approach, was used.

To the best of our knowledge, rare are the works which address jointly three constraints: the load-bearing, the weight limits, and the positioning. Then, in our work, we propose an approach that integrates all these constraints.

IV. THE PROPOSED PROCEDURE

A. Block generation

In this work, both simple blocks, that contain only boxes of the same type in one orientation, and General block, that contain multiple types of boxes in a different orientation, are addressed. A simple block is a replication of a given box in one orientation n_x, n_y and n_z time along the X, Y and Z axis (The container length, width, and height direction). The details of simple blocks generation are presented in Algorithm1.

In the orthogonal packing, there are six possible orientations for a rectangular box. But, this number could be restricted (for example the package of a refrigerator shall be upright), this restriction is due to the sheer stress and the fragility of some faces. Line 2 considers the possible orientations of one selected box. In a first step, the new block to be created has only one box along X as well as a length equal to the box length (line 3). If the container length and the maximal accepted number of boxes to be placed are not exceeded (line 4), then the number of boxes of the same type is equal to $n_x * n_y$, the block width is equal to the box width

(line 5). If it is allowed and the new block's width doesn't exceed the container width (line 6), the block will have a height equal to the box's height and only one box along Z. The generated block will be added to the list only if n_b (the number of boxes of a given boxes type) was not exceeded (line 9, 10, and 11). In a subsequent step (line 12, 13, and 14), the number of boxes along X, Y, and Z would be incremented, that is why the block height ($block_{Height}$) is incremented by the box height. It is the same approach for the block width and the length.

Algorithm 1: Generate simple blocks

Input: Box list

Output: Block list

1 **for** all boxes **do**
2 **for** all box orientations **do**
3 $n_x \leftarrow 1$; $block_{Length} \leftarrow box_{Length}$;
4 **while** ($n_x \leq n_b$ and $block_{Length} \leq container_{Length}$) **do**
5 $n_y \leftarrow 1$; $block_{Width} \leftarrow box_{Width}$;
6 **while** ($n_x * n_y \leq n_b$ and $block_{Width} \leq container_{Width}$) **do**
7 $n_z \leftarrow 1$; $block_{Height} \leftarrow box_{Height}$;
8 **while** ($n_x * n_y * n_z \leq n_b$ and $block_{Height} \leq Container_{Height}$) **do**
9 add the new block with $n_x * n_y * n_z$ along X, Y, and, Z to the Block list;
10 **if** (|Block list| = max_bl) **then**
11 **return** Block list;
12 $n_x \leftarrow n_x + 1$; $block_{Height} \leftarrow block_{Height} + box_{Height}$;
13 $n_y \leftarrow n_y + 1$; $block_{Width} \leftarrow block_{Width} + box_{Width}$;
14 $n_z \leftarrow n_z + 1$; $block_{Length} \leftarrow block_{Length} + box_{Length}$;

For general blocks, the same technique presented in [23] was used. In a first step, the block list contains only simple blocks. In a further step, two blocks are combined along the X, Y and the Z axis to create a larger one. Firstly, each box creates at the most six blocks that correspond to the six possible orthogonal orientations of a box. Iteratively, a combined procedure is invoked to combine them in contact along the axis. Only blocks that use min_fr% are accepted (volume of boxes over the volume of the block must be greater or equal to min_fr% generally equal to 98% of the volume of the bounding cuboid). Blocks with the same dimensions containing the same boxes are considered identical even if their internal configurations are different. Also, blocks containing more boxes of a certain type than what it is available are discarded and considered illegal configurations. Besides, blocks whose dimensions exceed the size of the container are not generated. The process stops when max_bl blocks are created or there are no more different cases (for more details the reader is referred to the algorithm of generating general blocks in [16]).

B. Free space representation

Generally, the free space in the container is represented as a set of rectangular parallelepipeds. On one hand, it is clear that the empty space in an initial stage is the container itself when no boxes have been loaded. On the other hand, when a block is loaded, the remaining free space is a polyhedron as shown in figure 1.

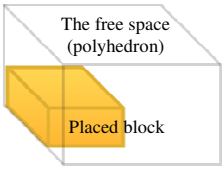

Fig. 1 The free space in the container

The maximal space representation, proposed by Lim et al. [24] is used. For each packed block, at most six cuboids are generated and cover each face of the loaded block. The resultant free space list contains overlapped parallelepipeds between each other but interior disjointed with the packed block. Figure 2 shows an example of only one place block (for a reason of clarity, the figure is illustrated in separate diagrams).

Fig. 2 The maximal space representation

The residual space in the container is presented as a linked list R= $\{r_1, r_2, \ldots, r_n\}$ of overlapped cuboids. When a block bi is loaded into a corner of the container, three flapped parallelepipeds are added to the free space stack. Similarly, each free space that intersects with any packed block is removed and up to 6 cuboids are added into R. Any space that couldn't contain any box is deleted and considered as a non-useful space (waste space). More details about updating R are presented in algorithm 2 and figure 3.

For each plane that corresponds to a face of the block b, if the plane intersects with r, then it will divide it into two parts, where one part overlaps with b and the other part pass through a residual space. Therefore, for any residual space that

overlaps with a placed block b, its remaining free space could be represented by up to 6 free spaces.

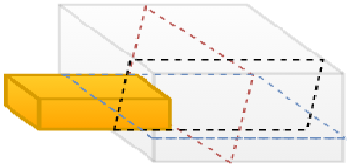

Fig. 3 Three possible generated free spaces for a block at the first corner

Algorithm 2: Update space list
Input: *block b, space s, state (volume utilization, free boxes, space list)*
Output: *space list*, state

```
if (the volume of b = the volume of s) then
    remove s from space list;
New space list ← Ø;
for all space s1 in the space list do
    if (b intersect s1 = Ø) then
        add s1 to New space list;
    Else
        dissect s1 in each intersection;
clear space list;
for all space s2 in new space list do
    add s2 to space list if it is legal;
```

It is legal to add a free space to the space list stack only if it could contain at least one box and shouldn't be totally located in another free space. Let's suppose that for a given space the corner closest to the origin has the coordinate (x_1,y_1,z_1) and the corner farthest from the origin has the coordinate (x_2,y_2,z_2). The algorithm that cuts a residual space to up six others at the most is described as Algorithm3 below and Figure 4 presents more details:

Algorithm 3: Dissect
Input: space s, block b, space list
Output: *space list*

```
if (s.x1 ≠ b.x1) then
    add the space (s.x1,s.y1,s.z1,b.x1,s.y2,s.z2);
if (s.x2 ≠ b.x2) then
    add the space (b.x2,s.y1,s.z1,s.x2,s.y2,s.z2);
if (s.y1 ≠ b.y1) then
    add the space (s.x1,s.y1,s.z1,s.x2,b.y1,s.z2);
if (s.y2 ≠ b.y2) then
    add the space (s.x1,b.y2,s.z1,s.x2,s.y2,s.z2);
if (s.z1 ≠ b.z1) then
    add the space (s.x1,s.y1,s.z1,s.x2,s.y2,b.z1);
if (s.z2 ≠ b.z2) then
    add the space (s.x1,s.y1,b.z2,b.x1,s.y2,s.z2);
```

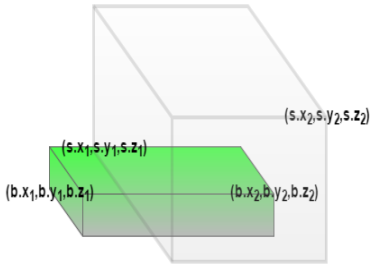

Fig.4: A placed block b overlaps with a residual space

C. Search state and transition

Our approach works by exploring the search space. It is a tree search starting from a root node where no blocks are loaded, the free space is the container itself, all the boxes remain not placed, and the block list holds all the generated blocks. The transition from state to another occurs when placing a block in a free space. For each loaded cuboids, the list of remaining boxes is updated by discarding them. The blocks containing more boxes than is available are also deleted since there are fewer left. After placing a block into an empty space at one of its corners, the cover representation will represent the remaining as a list of overlapped cuboids; this representation is used in many successful algorithms ([16]-[18]). Afterward, we update the free space list (see algorithms 2 and 3) by removing the cuboids that intersect with the loaded blocks and generating up to six new ones. When there are blocks and free spaces in a state, we select a free space that minimizes the Manhattan distance: for the eight corresponding corner pairs, we calculate the distance $|x_1-x_2| + |y_1-y_2| + |z_1-z_2|$ to the container. The corner with the smallest value is the anchor corner of s, and the distance between the anchor corner and its corresponding corner of the container is the anchor distance. The free space that minimizes the Manhattan distance is selected to be filled. If all the available blocks can't fit any free space, this state represents a terminal state and the corresponding loading plan is a maximal packing.

D. The Overall Approach (OAA)

We propose a greedy two-step look-ahead procedure by selecting a free space deterministically followed by a block search. More precisely, the cover representation is used to represent the free space, since it does not restrict the search space to only guillotine cuts. Then, we construct simple blocks for weakly heterogeneous problem instances, and general blocks for strongly heterogeneous problem instances.

We select a free space from the space list stack that minimizes the Manhattan distance. The purpose behind this choice is to first pack closer to the corner, then the sides and then the faces of the container. The space in the container tends to be continuous and the fragmentation would be reduced. The selection of a block is based on a two-step look-ahead search approach with an appropriate fitness function. If a block is selected, it is packed at the anchor corner. Finally, the search effort is doubled at each iteration of the global procedure. Algorithm 4 summarizes the overall approach.

Algorithm 4: OAA procedure
Input: *Box list, Container*
Output: *Best Solution*

```
Initialize and get data;
Evaluate heterogeneity;
if (weakly heterogeneous case)
    generate simple blocks;
else
    generate general blocks;
Best Solution ← Ø;
Search effort ← 1;
while (the consumed CPU time is not exceeded) do
    w ← integer part (√(Search effort)) ;
    if (there is a redundancy) then
        Search effort ← 2*Search effort;
        continue;
    the free space list contains the container itself;
    while (the free space list is not empty) do
        Compare the free spaces then select the best one;
        Perform a two-depth tree search by ranking w
        blocks according to the fitness function for each
        node then conserve the most promising one;
        if (there is at least one block) then
            Pack b at the anchor corner;
            Update block list;
            Update free space list;
        else delete the free space since no block can fit it;
            Search effort ← 2*Search effort;
Return the best solution;
```

E. Approach generalization

In a first stage, the classical single container loading problem has been investigated and an approximate procedure was proposed. In a second stage, we have adapted the overall approach in order to fit the weight distribution, the positioning, and the stacking constraints to deal with realistic situations. In fact, during transportation, the weight distribution reduces the risk of boxes shifting when the cargo is moved. The gravity center must be as close as possible to the container floor midpoint. Then, the aim of the positioning constraint is the restriction of certain locations in the container for some items such as the package of volatile liquids and explosive products that should be as close as possible to the top of the container, thus, they could be accessed and removed quickly, if necessary. Finally, the stacking constraint, also called the load-bearing constraint, restricts the placement of boxes on top of each other and imposes that some items could only support a limit weight or pressure. Therefore, some box orientations are restricted depend on the load-bearing strength. Besides, the placement of cuboids on top of each other relies on the fragility classification: non-fragile boxes could only be supported by other non-fragile boxes, but not on the fragile ones.

V. COMPUTATIONAL RESULTS

In order to evaluate its computational performance, the proposed procedure is implemented using the 64-bit Java development kit 1.7.0 from Eclipse. All the computational experiments were carried out on an i5 dual core 2.2 GHz Personal Computer with 8.0 GB RAM.

A. The classical single container loading problem

To test the effectiveness of our approach without additional constraints, we use the standard test instances of Bishoff and Raticliff [10]. More precisely, there are over than 1600 Benchmark instances divided into 16 test files/sets of 100 instances each, named BR0–BR15 and available in the OR library [25]. They are commonly used in the literature and are classified into three categories: BR0: homogenous; BR1–BR7: weakly heterogonous; and BR8–BR15: Strongly heterogonous.

For each instance among the 1600 instances, we create a file that contains the position of each packed box related with some other necessary data (for example: the box type which is used to select a color for the box when it is 3D drawn…). For the comparison with other approaches, we collect all the volume utilization indicators, calculate the average and compare them with the other existing approaches in a next step.

The results of the overall procedure proposed in algorithm 4 are summarized in Table I: we present for each test set BRi, i=0,..,15, the corresponding total type of boxes, the average of the volume utilization in percentage and the average computing CPU time in seconds.

TABLE I
PERFORMANCE OF THE PROPOSED OAA

Test Set	*Total type of boxes*	*Average volume utilization*	*Average CPU Time*
BR0	1	90.70	36.50
BR1	3	95.37	50.00
BR2	5	95.74	50.00
BR3	8	95.82	30.00
BR4	10	95.55	30.00
BR5	12	95.38	30.00
BR6	15	95.20	30.00
BR7	20	94.71	30.00
BR8	30	94.07	30.01
BR9	40	93.49	30.59
BR10	50	93.21	30.01
BR11	60	92.79	30.55
BR12	70	92.73	41.11
BR13	80	92.73	52.31
BR14	90	92.60	59.36
BR15	100	92.69	61.31
Average	*37.13*	***93.92***	*38.86*

Based on figure 5 derived from Table I, we can mention that the OAA generates more efficient solutions for weakly heterogeneous instances than the strongly heterogeneous instances since these last ones are the most difficult to solve.

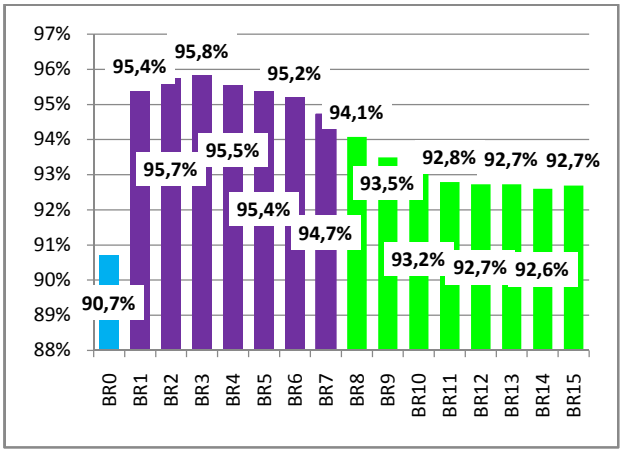

Fig. 5: The average volume utilization

The performance comparison between our approach and the most known heuristics from the literature, in terms of volume utilization, is summarized in Table II. The column headings are as follows: ***GRASP***: Greedy Randomized Adaptive Search Procedure [14], ***FDA***: Fit Degree Algorithm [26], ***VNS***: Variable Neighborhood Search [8], ***TRS***: Tree Search [23], ***G2LA***: Greedy 2-Step Lookahead [16], ***IDGLA***: Iterative-Doubling Greedy–Lookahead [18], ***BS***: Beam Search [19].

TABLE II
COMPARISON OF THE PROPOSED PROCEDURE WITH THE MOST KNOWN HEURISTICS

Test Set	*GRASP*	*FDA*	*VNS*	*TRS*	*G2LA*	*IDGLA*	*BS*	*OOA*
BR0	-	-	-	90.0	90.8	90.8	91.0	90.7
BR1	89.1	92.9	94.9	95.1	95.5	95.5	95.7	95.4
BR2	90.4	93.9	95.2	95.4	96.0	96.1	96.2	95.7
BR3	90.9	93.7	95.0	95.5	96.1	96.2	96.5	95.8
BR4	90.4	93.7	94.7	95.2	95.9	96.1	96.3	95.5
BR5	89.6	93.7	94.3	95.0	95.7	95.9	96.2	95.4
BR6	89.7	93.6	94.0	94.8	95.6	95.7	96.6	95.2
BR7	88.1	93.1	93.5	94.2	95.1	95.3	95.8	94.7
BR8	86.1	92.9	92.8	93.7	94.6	94.7	95.3	94.1
BR9	85.1	92.5	92.2	93.4	94.3	94.5	95.1	93.5
BR10	84.2	92.2	91.9	93.1	94.0	94.3	95.0	93.2
BR11	84.0	91.9	91.5	92.8	93.8	94.1	94.8	92.8
BR12	83.6	91.8	91.2	92.7	93.7	94.0	94.6	92.7
BR13	83.5	91.6	91.1	92.5	93.5	93.8	94.6	92.7
BR14	83.3	91.3	90.6	92.4	93.4	93.8	94.5	92.6
BR15	83.2	91.0	90.4	92.4	93.3	93.7	94.4	92.7
Average	***86.7***	***92.7***	***92.9***	***93.9***	***94.7***	***94.9***	***95.1***	***93.9***
Average CPUTime	***69***	***633***	***296***	***320***	***500***	***303***	***500***	***39***

It is noteworthy to note that the computing CPU Times could not be compared directly because in each work a specific computing machine was used. Obviously, for all approaches, the running times are lower for the weakly heterogeneous problems and increase with the number of different box types. Our approach outperforms all the existing approaches that were published before 2012 while taking much less CPU Time computing. As described in section II, the objective of this work is to develop a resolution procedure providing good results within very reasonable times.

B. Considering the weight distribution constraints

The closer the gravity center is to the middle of the length and width, the better the down height is. The evaluating criterion of weight distribution is defined using the center of gravity of loaded boxes; while for each box with an even weight density, the center of gravity of the box coincides with its geometric center. The nearer the center of gravity is to the ideal center of gravity for the container, the evener the weight distribution of boxes is.

To test the effectiveness of our algorithm when fitting the weight distribution constraints, we use the test data sets named wtpack1–wtpack7 [27], and we set the maximum CPU time to 50 seconds for all the following computational tests. Table III summarizes the OAA results showing that an average volume utilization of more than 90% is provided.

TABLE IIII
PERFORMANCE OF THE OAA WHEN CONSIDERING THE WEIGHT DISTRIBUTION CONSTRAINTS

Test Set	*Average volume utilization*
wtpack1	90.48
wtpack2	90.56
wtpack3	90.99
wtpack4	90.76
wtpack5	90.83
wtpack6	91.00
wtpack7	90.59
Average	***90.74***

C. Considering all the additional constraints

In this section, we consider simultaneously the weight distribution, positioning, Load-bearing constraints. To the best of our knowledge, there is no work in the literature dealing with such realistic problem. Thus, we create 21 test instances, which are the 3 first instances from the wtpack test set. For each box line type, two additional binary parameters are added. The first parameter indicated whether the boxes in this line shall be placed on top or not. It takes1, if we shall place this kind of boxes on top, 0 otherwise. The second binary parameter indicates if a box is fragile or not. It is equal to 1 if the box is fragile, 0 otherwise. We compromise with the positioning but the load-bearing is restrictive (no non-fragile box is placed on top of a fragile box). The results are presented in Table IV. The first column holds the label of the test pack: we call the new set data SPWpack (S stands for stacking, P for positioning and W for weight distribution). The integer after SPWpack determines the number of box types. The second column indicates the total volume utilization in percentage.

TABLE IV
PERFORMANCE OF THE OAA WHEN CONSIDERING ALL THE ADDITIONAL CONSTRAINTS

Test Set	*Average volume utilization*
SPWpack3	90.81
SPWpack5	89.22
SPWpack8	90.11
SPWpack10	89.81
SPWpack12	90.26
SPWpack15	90.70
SPWpack20	89.60
Average	***90.07***

From Table IV, we can conclude that even if we take into consideration all the additional realistic constraints, the tailored procedure still provides, in average, solutions with up to 90% of volume utilization in at most 50 seconds.

VI. CONCLUSION

We investigate the Single Container Loading Problem, the problem of loading rectangular boxes into containers, which is a common basic activity in material handling. In this work, we propose a new heuristic for the single container loading problem. The aim is to pack three-dimensional small items called boxes in a three-dimensional big item called container efficiently so as the volume utilization is maximized. The free space selection and the block placement are performed deterministically, while a greedy-two-step lookahead algorithm ranks a block for a select empty space. The overall approach was developed using the Java language and extensively tested on over than 1600 Benchmark instances from the literature. The computational experimentation proves that the proposed procedure is competitive, as good solutions are found in very reasonable times. To go further in taking into consideration more realistic constraints, we adapted our algorithm to fit the weight distribution, the positioning, and the stacking constraints. The final framework would be used to complete the 3D graphic in order to implement it in the EWM related module of SAP.

ACKNOWLEDGMENT

The authors would express their gratitude to the personal of "Carthago Solutions Company" and more precisely Achref ZAIDI for his kind and valuable input towards this work.

REFERENCES

[1] (2015) The 2014 market review. ALPHALINER. [Online]. Available: http://www.alphaliner.com/liner2/research_files/newsletters/2015/no03/Alphaliner%20Newsletter%20no%2003%20-%202015.pdf

[2] (2016) SAP, n.d. Supply Chain Management/ SAP Extended Warehouse Management. [Online] Available: http://go.sap.com/product/scm/extended-warehouse-management.html

[3] D. Pisinger, "Heuristics for the container loading problem", *European Journal of Operational Research*, vol. 141, pp. 382-92, Sep. 2002.

[4] S. P. Fekete, J. Schepers, and J. C. van der Veen, "An exact algorithm for higher-dimensional orthogonal packing", *Operations Research*, vol. 55, pp.569–587, Jun. 2007.

[5] L. Junqueira, R. Morabito , and D. S. Yamashita, "Three-dimensional container loading models with cargo stability and load bearing constraints", *Computers & Operations Research*, vol. 39, pp. 74–85, Jan. 2012.

[6] C. F. Chien, and W. T. Wu, "A recursive computational procedure for container loading", *Computers & industrial engineering*, vol. 35, pp. 319-322, Oct. 1998.

[7] L. Lins, S. Lins, and R. Morabito, "An n-tet graph approach for non-guillotine packings of n-dimensional boxes into an n-container", *European Journal of Operational Research*, vol. 141, pp. 421-439, Sep. 2002.

[8] F. Parreño, R. Alvarez-Valdés, J. F. Oliveira, and J. M. Tamarit, "Neighborhood structures for the container loading problem: a VNS implementation", *Journal of Heuristics*, vol. 16, pp. 1-22, Feb. 2010.

[9] J. A. George, and D. F. Robinson, "A heuristic for packing boxes into a container", *Computers & Operations Research*, vol. 7, pp. 147-156, Dec. 1980.

[10] E. E. Bischoff, and M. S. W. Ratcliff, "Issues in the development of approaches to container loading", *Omega*, vol. 23, pp. 377-390, Aug. 1995.

[11] H. Gehring, and A. Bortfeldt, "A genetic algorithm for solving the container loading problem", *International Transactions in Operational Research*, vol. 4, pp. 401-418, Nov. 1997.

[12] A. Bortfeldt, and H. Gehring, "A hybrid genetic algorithm for the container loading problem", *European Journal of Operational Research*, vol. 131, pp. 143-161, May. 2001.

[13] A. Bortfeldt, H. Gehring, and D. Mack, "A parallel tabu search algorithm for solving the container loading problem", *Parallel Computing*, vol. 29, pp. 641-662, May. 2003.

[14] A. Moura, and J. F Oliveira, "A GRASP approach to the container-loading problem," *IEEE Intelligent Systems,* vol. 20, pp.50–57, Jul. 2005.

[15] Z. Jin, K. Ohno, and J. Du, "An efficient approach for the three-dimensional container packing problem with practical constraints", *Asia-Pacific Journal of Operational Research*, vol. 21, pp. 279-295, Sep. 2004.

[16] W. Zhu, W. C. Oon, A. Lim, and Y. Weng, "The six elements to block-building approaches for the single container loading problem," *Applied Intelligence*, vol. 37, pp. 431–445, Oct. 2012.

[17] F. Parreño, R. Alvarez-Valdés, J. M. Tamarit, and J. F. Oliveira, "A maximal-space algorithm for the container loading problem," *INFORMS Journal on Computing,* vol. 20, pp. 412–422, Aug. 2008.

[18] W. Zhu, and A. Lim, "A new iterative-doubling Greedy–Lookahead algorithm for the single container loading problem", *European Journal of Operational Research*, vol. 222, pp. 408-417, Nov. 2012.

[19] I. Araya, and M. C. Riff, "A beam search approach to the container loading problem", *Computers & Operations Research*, vol. 43, pp. 100-107, Mar. 2014.

[20] J. Liu, Y. Yue, Z. Dong, C. Maple, and M. Keech, "A novel hybrid tabu search approach to container loading", *Computers & Operations Research*, vol. 38, pp. 797-807, Apr. 2011.

[21] N. Wang, A. Lim, and W. Zhu, "A multi-round partial beam search approach for the single container loading problem with shipment priority", *International Journal of Production Economics*, vol. 145, pp. 531-540, Oct. 2013.

[22] A. Lim, H. Ma, C. Qiu, and W. Zhu, "The single container loading problem with axle weight constraints", *International Journal of Production Economics*, vol. 144, pp. 358-369, Jul. 2013.

[23] T. Fanslau, and A. Bortfeldt, "A tree search algorithm for solving the container loading problem," *INFORMS Journal on Computing,* vol. 22, pp. 222–235, May. 2010.

[24] A. Lim, B. Rodrigues, and Y. Wang, "A multi-faced buildup algorithm for three-dimensional packing problems", *Omega*, vol. 31, pp. 471-481, Dec. 2003.

[25] (1990) The OR Library. [Online]. Available: http://people.brunel.ac.uk/~mastjjb/jeb/info.html

[26] K. He, and W. Huang, "An efficient placement heuristic for three-dimensional rectangular packing," *Computers & Operations Research,* vol. 38, pp. 227–233, Jan. 2011.

[27] (1998) The OR Library: Container Loading with weight restrictions. [Online]. Available: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/conloadinfo.htm