A simulated annealing algorithm for single container loading problem

Wang Hongtao, Wang Zhoujing, Luo Jian
Department of Automation
Xiamen University
Xiamen, China
wanghongtao.2006@yahoo.com.cn

***Abstract*—A three-dimensional single container loading problem (3D-CLP) is aimed to load kinds of rectangular boxes into a single container with maximal volume utilization. This paper presents a multi-stage search based simulated annealing algorithm (MSSA) for single container loading problem (3D-CLP). An approach for representation of feasible solution is presented. An empty maximal-space list is used to manage the free spaces and difference process is applied in space decomposition. For simulated annealing algorithm is an individual improve approach which rely on a large number of trials, a multi-stage search process is applied to improve accuracy. This approach is tested on the set of test cases proposed by Bischoff and Ratcliff [1], and a comparative result is presented.**

Keywords-container loading problem; simulated annealing; heuristics; multi-stage

I. INTRODUCTION

The efficient use of transport capacities is getting more and more important in production and distribution processes. The effort to transport more goods/items with low cost, less energy and time introduces the question of the effective use of containers. As a result, the container loading problem (CLP) which is aimed at packing goods optimally or at least near optimally becomes an active research area. The CLPs can be categorized as homogeneous case and heterogeneous case. The heterogeneous CLP, in a basic form, can be characterized as follows: arranging a set of rectangular items (boxes) into one rectangular large object (container) in such a way that the total volume of the packed items is maximized (also defined as 3/B/O problem in [2]). Since CLP is notoriously NP-hard, the majority of work in recent years is focused on heuristics methods.

According to Fanslau and Bortfeldt [3], heuristic method without an optimality guarantee for 3D-CLP can be divided into conventional heuristics, meta-heuristics and tree search methods. Conventional heuristics include construction methods and improvement methods，and were suggested, e.g., in papers by Bischoff at al [4], Bischoff and Ratcliff [1], and Lim at all [5]. Meta-heuristics include intelligent algorithms such as genetic algorithms (e.g. Gehring and Bortfeldt [6], Bortfeldt and Gehring [7] and Gonçalves and Resende [8]), simulated annealing algorithms (Mack et al. [9]), tabu search (Bortfeldt at al [10]), the approach of local search by Bischoff [11] and the greedy randomized adaptive search procedures of Moura and Oliveira [12]. Tree-search methods have been suggested among others by Morabito and Arenales [13], Eley [14], Hifi [15], Pisinger [16] and Fanslau and Bortfeldt [3].

A simulated annealing algorithm for single container loading problem is presented in this paper and it includes two main parts, the presentation and generation of feasible solution and the evolution of solution. A feasible solution is encoded as box packing order sequence, layer type sequence and layer position sequence. An empty maximal-space list is used to manage the free spaces. Difference process, developed by Lai and Chan [17], is applied in space decomposition. A new search method called multi-stage search is used in the evolution of solutions. When an equilibrium state is approached, the traditional simulated annealing chooses the last accepted solution as the starting point of next iteration, which, however, is always not a good choice. For the reason that the final result of iteration depends much on the starting point, multi-stage search make efforts to find a good starting point. A multi-stage search consists of several stages, and the number of stages is dynamically determined according whether a better solution is found.

The following sections are organized as: in Section 2, the problem addressed by this approach is expressed. Section 3 explains the representation and the generation of feasible solution. In the beginning of Section 4, the generic aspect of simulated annealing are described, and the remainder of it gives the details of this approach. Comparative studies with other algorithms and an illustrative example are given in Section 5 and some concluding remarks are provided in Section 6.

II. PROBLEM STATEMENT

This paper assumes that any mix types of cargos which packed in rectangular boxes should be loaded into a single standard container; the types of boxes can range from weakly to strongly heterogeneous. All these rectangular boxes are packed feasible with the objective of maximizing the packed volume. This paper disregard the constraints proposed by Bischoff and Ratcliff [1], and only following basic principles are met:

(1) Each box must be completely packed within container;

(2) There no overlap between any two boxes;

978-1-4577-2025-3/12/$26.00 ©2012 IEEE

(3) Box can be rotated in six possible orientations and should be parallel to the edges of the container.

III. Representation And Generation of Feasible Solutions

A. Representation of feasible solution

A feasible solution is expressed as three separate sequences: box type packing order sequence, layer type sequence and layer position sequence.

1) Box type packing order sequence: For a given N box types and each type contains n_k cuboids, there are a total of $M = \sum_{k=1}^{N} n_k$ boxes. The box type packing order sequence (BTPS) is made of M elements which represent the types of packing box. Box should be packed by the order of BTPS. For example, the sequence 2, 4, 1, 2, 3 means we should first pack boxes of type 2, then type 4, and so on. For a type of box, this approach pack boxes in a layer. A layer is a rectangle which is composes of boxes of same type in rows and columns, with the depth of one.

2) Layer type sequence (LTS): Layer type sequence has the same length to box packing order sequence. Layer type is an integer not less than 1 and not greater than 36. Every element in the box packing order sequence is mapping to a layer type in layer type sequence. Layer type contains the message that directs how to arrange boxes of the same size into a layer. Each box type can has at most six rotation variants. For each rotation variant, we can arrange boxes into a layer in six means (See Fig. 1). Therefore, we get 36 layer types totally [20].

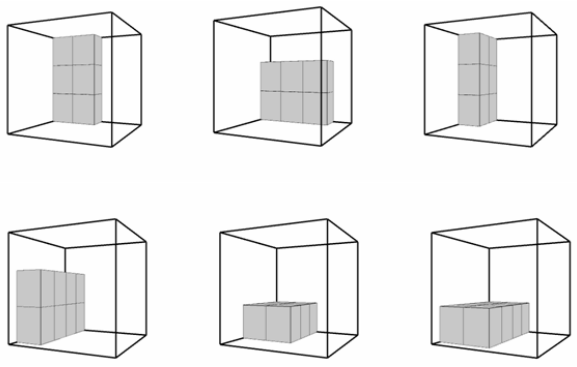

Figure 1. Example of the six different feasible layer types for a box type rotation variant with at most 8 boxes.

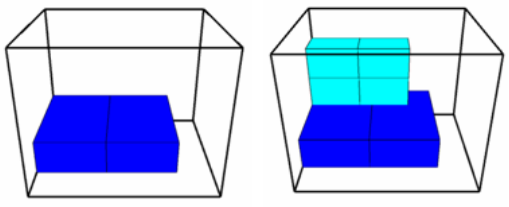

Figure 2. Layer packing illustration.

3) Layer position sequence (LPS) records the locations, dimensions of layers and numbers of boxes contained in layers. LPS has same size of BTPS. The kth element in a typical LPS has the form of $[(x_k, y_k, z_k), (l_k, w_k, h_k), n_k]$, where (x_k, y_k, z_k) is the minimum coordinate of layer in container, (l_k, w_k, h_k) is the dimensions of layer, and n_k is the number of boxes layer contains.

To demonstrate the relation between BTPS, LTS and LPS, suppose the minimum coordinate of container is (0, 0 ,0), given a BTPS: {2, 2, 1,…}, a LTS: {34, 12, 5, …}, and a LPS: {[(0, 0, 0), (30, 40, 10), 2], [(0, 0, 0), (0, 0, 0), 0], [(0,0,10),(10,32,20),4],…}, that is, 2 boxes of type 2 are arranged to a layer of 34, and packed into container so that the minimum coordinate of layer in container is (0,0,0), and dimensions of this layer is (30,40,10); The second layer (empty layer) does not contain any box (all elements in LPS[2] are zero). 4 box of type 1 are arranged to a layer of type 5 so that the minimum coordinate of third layer is (0, 0, 10) and its dimensions is (10, 32, 20) (see Fig. 2).

B. Generation of feasible solution

1) Empty maximal-space and space decomposition: A list of empty maximal-spaces (EMSs) [20] is a list of spaces which are available when packing boxes. Before packing, the space of empty container is added to EMSs which is available for boxes. The list of EMSs are ordered by their minimum coordinates(x, y, z) in such a way that EMSi<EMSj if xi<xj, or if xi=xj and zi<zj, or if xi=xj, zi=zj, and yi<yj (back-bottom-left procedure). When try to pack a layer, the first EMS which at least a box of current type to be packed fits in is chosen and is erased from EMSs after packing, new empty available spaces generated are added to EMSs. When packing a layer into an EMS, we use the difference process, developed by Lai and Chan [17], to generate new empty available spaces. We denote a rectangular by its back bottom left (minimum) coordinate and front upper right (maximum) coordinate. Assume that x1< x2, x3<x4, y1<y2, y3<y4，z1<z2 and z3<z4, the result of cutting a rectangular [(x3, y3, z3), (x4, y4, z4)] into a rectangular [(x1, y1, z1), (x2, y2, z2)] can be expressed as:

$$
\begin{aligned}
&[(x1, y1, z1), (x2, y2, z2)] - [(x3, y3, z3), (x4, y4, z4)] = \\
&\{ \\
&\quad valid[(x1, y1, z1), (x3, y2, z2)], \\
&\quad valid[(x4, y1, z1), (x2, y2, z2)], \\
&\quad valid[(x1, y1, z1), (x2, y3, z2)], \\
&\quad valid[(x1, y4, z1), (x2, y2, z2)], \\
&\quad valid[(x1, y1, z1), (x2, y2, z3)], \\
&\quad valid[(x1, y1, z4), (x2, y2, z2)] \\
&\}.
\end{aligned}
\tag{1}
$$

Where

$$valid[(xm,ym,zm),(xn,ym,zn)] = \begin{cases} null, if\ xm \geq xn\ or\ ym \geq yn\ or\ zm \geq zn, \\ [(xm,ym,zm),(xn,ym,zn)], others. \end{cases} \quad (2)$$

The new generated spaces are added to *EMSs* which are available for next layer.

2) The packing process contains the following steps*:*

Step1. Box type and layer type selection. Get the current packing box type N from BTPS, layer type K from LTS;

Step2. EMS selection. Choose a first fit empty maximal-space (EMS) from EMSs to pack the new box layer specified by N and K. If the EMS is too small to contain a box, skip to choose next EMS. Difference process is applied when cutting away the layer from EMS to produced new available spaces.

Step3. State information update.

(1) Update the remaining quantities of the box type N;

(2) Update available spaces list EMSs.

(3) Record the position of layer into LPS.

Step4. If all boxes are packed or there is no EMS which any box can fits, finish packing, otherwise, go step 1.

IV. AN IMPROVED SIMULATED ANNEALING ALGORITHM

A. Simulated annealing review

Simulated annealing (V. Cerny [18]) is a Monte Carlo algorithm to search approximate solutions of a combinatorial optimal problem. Simulated annealing is a general-purpose optimization procedure, it concerns two main concepts: configuration and objective function. Every configuration represents a feasible solution and the objective function measures the performance of the solution.

The searching process start with an initial solution which represented by a configuration, and then we calculate the value of objective function. After a neighborhood search, if a better solution is found, the new configuration replaces the old one. Otherwise, the new one is accepted with probability $e^{-\Delta C/T}$ (ΔC is the reduction of the performance, T is positive control parameter called temperature which decrease by time according to a defined cooling schedule). In other words, a higher cost and a lower temperature would lead to a lower acceptance probability. The process repeat neighbor search until some condition is met or no further improvement can be found.

B. Neighborhood search

A feasible solution can be confirmed by BPTS and LTS. This simulated annealing algorithm keeps neighborhood searching from current solution. A new feasible solution should be close to the old one [18], and, therefore, is generated by two means (See Fig. 3) : (1) Exchanging two elements of box packing order sequence; (2) Modify one element of layer type sequence.

C. A multi-stage search process

This algorithm tries to find a good placement pattern in all possible solutions. As we known, the temperature in annealing simulated can control the possibility of acceptance of worse solution to prevent premature convergence. In the earlier rounds, a relatively high temperature is needed to conduct trials widely, and in the latest round, the temperature is low enough to get a precise local optimal solution. Unlike traditional annealing simulated process which is more like a link search, a multi-stage search processes the search procedure in several search stages. In each stage, a link search as traditional way is process. The best solution in this stage which also is the best solution in a link search is chosen to be the start of next stage. The pseudo-code of the simulated annealing algorithm is given in Fig. 4.

A search process in a fixed temperature contains several stages. If a new best solution was found in a search stage, which indicates that it is potential to reach a better equilibrium in current temperature, recount the stage search number and save the best solution to be the start of next search stage (lines 23-25 of the pseudo-code). If there no any improvement in *nl* search stages, which means the equilibrium is reached, cool down the temperature and start to find the new equilibrium (line 32 of the pseudo-code). The fitness value of a solution specified by BTPS and LTS is calculated by function *calculateFitness*. A worse solution is accepted occasional according to the temperature (lines 26-28 of the pseudo-code).

D. Cooling schedule

The chosen cooling schedule has an important effect on the performance of the simulated annealing algorithm. In our study, the initial temperature T_0 is set to 0.064 for the reason that the possibility of acceptance of worse solution is $p_0 = e^{(-\Delta f/T_0)}$, where Δf is the reduce of fitness value, and in the temperature of T_0 a worse solution with the fitness about 5% less than the previous solution is expected to be accept with 90% of the probability. The k *th* temperature $T_{k-1} = T_{k-2}/1.6$.

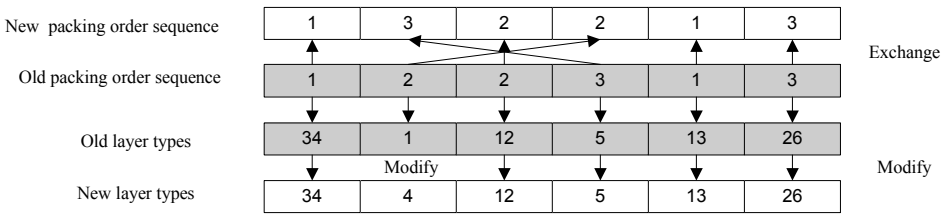

Figure 3. Example how to get a neighborhood placement pattern.

```
bestFitness = Procedure annealing()
1   Initialize the start placement pattern randomly which consist of BTPS, LTS;
2   (fitness, LPS )= calculateFitness(BTPS, LTS); // ** Calculate the fitness and record the layer positions
3   Let ni be the number of times the temperature changes, let nl be the number of stages, let nt be the
            number of rounds, initialize temperature with the initial temperature T;
4   Let bestFitness = fitness, bestBTPS = BTPS, bestLTS = LTS,
            bestLPS = LPS;  // ** To save the best solution

5   for i:= 0 to ni :
6     for j:=0 to nl :  // ** Search in stages
        // ** Use best solution as the start of next stage
7       Let fitness = bestFiness, BTPS = bestBTPS , LTS = bestLTS, LPS=bestLPS;

8       for k:= 0 to nt : // ** Search in rounds
9         for p:=0 to ALEN (length of LPS with the trailing empty elements removed):// ** Search a round
10          if (LPS[p].n) = 0 then // ** The pth layer recorded by LPS[p] contains no box.
11            continue;
12          end if
            // ** Get a neighborhood solution
13          Let nextBTPS = BTPS, nextLTS = LTS;
14          if rand(0, 1) < 0.1 then //** Exchange
15            Let q be a random integer between p (exclude) and length of BTPS (exclude);
16            Let nextBTPS[p] = BTPS[q],  nextBTPS[q] = BTPS[p];
17          else then //** Mutate
18            Let nextLTS [p] be a new integer range from 1 to 36 ;
19          end if
20          Let (nextFitness, nextLPS) = calculateFitness(nextBTPS, nextLTS );
21          if nextFitness> fitness then
22            fitness = nextFitness, LTS = nextLTS, BTPS = nextBTPS, LPS = nextLPS;

              // ** Save best solution
23            if nextFitness > bestFitness then
24              bestFitness = nextFitness, bestLTS = nextLTS , bestBTPS = nextBTPS,
                      bestLPS = nextLPS, j=0 ; // ** Recount j
25            end if

26          else if rand(0,1) < exp((nextFitness-fitness)/temperature) then
27            fitness = nextFitness, LTS = nextLTS, BTPS = nextBTPS, LPS = nextLPS;
28          end if
29        end for  // **  End a search round

30      end for  // ** End a search stage
31    end for
32    updateTemperature(); // ** Update temperature
33  end for
34  return bestFitness;
end annealing
```

Figure 4. Pseudo-code of the simulated annealing algorithm.

E. Fitness function

A fitness function is defined to measure the performance of a feasible solution. For container loading problem, the concern is the percent total packed volume, that is,

$$\frac{\sum_{k=1}^{N} v_k}{L \times W \times H}. \qquad (3)$$

Where v_k is the volume of kth packing layer recorded in LPS[k], L , W and H are dimensions of the rectangular container.

When process neighborhood searching, fitness function measure the performance of neighborhood solution to decide if this new solution would replace the old one.

V. NUMERICAL EXPERIMENTS

In this section, we use the benchmark problems generated and used in Bischoff and Ratcliff [1]. These problems are divided into 7 test cases and each contains 100 instances with same number of kinds of cargo. These test cases referred to as BR1 to BR7 contain 3, 5, 8, 10, 12, 15 or 20 different box types and the aim is to packing kinds of boxes into a single container to maximize the volume utilization for each problem.

A. Configurations

To get the ideal values of *ni, nl, nt*, we conduct some independent runs of algorithm. The values shown in table 2 are decided finally with the time and performance trade-offs considered.

TABLE I. PERFORMANCE COMPARISON OF ALGORITHMS WHEN SUPPORT CONSTRAINS IS NOT ENFORCED.

Volume Uilization								
Test case	*Lim et al. [5]*	*Betfekdt et al.[19]*	*Parreno et al.[20]*	*Parreno et al.[21]*	*Mack et al.[9]*	*Fanslau and Bortfeldt[3]*	*Gonçalves and Resende[8]*	*MSSA (this work)*
BR1	88.00	93.52	93.85	94.93	93.70	95.05	95.28	95.51
BR2	88.17	93.77	94.22	95.19	94.30	95.43	95.90	95.63
BR3	87.52	93.58	94.25	94.99	94.54	95.47	96.13	95.59
BR4	87.58	93.05	94.09	94.71	94.27	95.18	96.01	95.55
BR5	87.30	92.34	93.87	94.33	93.83	95.00	95.84	95.21
BR6	86.86	91.72	93.52	94.04	93.34	94.79	95.72	94.89
BR7	87.15	90.55	92.94	93.53	92.50	94.24	95.29	94.30
Average	87.61	92.70	93.82	94.53	93.78	95.02	95.74	95.24

TABLE II. THE VALUES OF PARAMETERS

Parameter	*Value*
ni	12
nl	10
nt	200
T	0.064

B. *Comparisons and example*

In the table 1, test result of this simulated annealing algorithm is presented. To demonstrate the performance of this algorithm, results from other approaches when support constrain is not enforced are used for comparison. A problem from test case BR7 is illustrated in Fig. 5 and Fig. 6.

VI. CONCLUSIONS

This paper presents an SA algorithm for solving the CLP with a single container to be loaded. The information of a feasible solution is encoded as three sequences: box packing order sequence, layer type sequence and Layer position sequence. A multi-stage search method is proposed to reduce huge number of the trails which are required in traditional simulated annealing algorithm and the accuracy is improved. An approach for representation of feasible solution is presented. The temperature is cooling down until some condition is met to dynamically control the number of iteration of trails. 700 CLP problems are tested to demonstrate the performance of this approach.

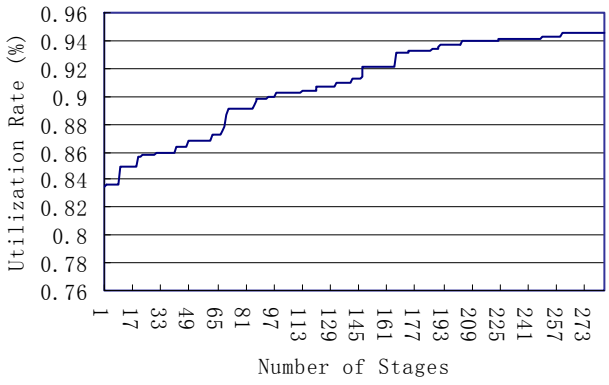

Figure 5. Convergence graph of a problem from test case BR7.

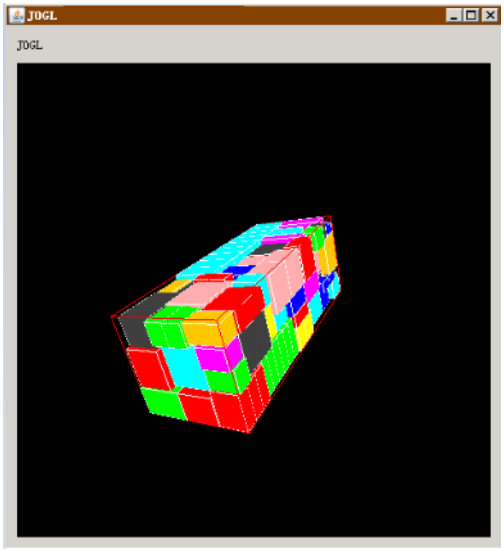

Figure 6. A packing view of a problem from test case BR7.

During the experiment, one shortcoming of this algorithm is also found that the results of some cases are not good enough compared with the average value. For only an individual is involved in simulated annealing, the chosen of solution which is the entry of next state (after temperature is updated) is required to be good enough or else some worse result is obtained. For further consideration, some combination of SA and GA may be useful to avoid the shortcoming.

REFERENCES

[1] E. E. Bischoff and M. S. W. Ratcliff, "Issues in the development of approaches to container loading," International Journal of Management Science, Omega, vol. 23, pp. 377–390, 1995.

[2] H. Dyckhoff, "A typology of cutting and packing problems," European Journal of Operational Research, vol. 44, pp. 145−159, January, 1990.

[3] T. Fanslau and A. Bortfeldt, "A tree search algorithm for solving the container loading problem," INFORMS Journal on Computing, vol. 22, pp. 220-235, 2010.

[4] E. E. Bischoff, F. Janetz and M. S. W. Ratcliff, "Loading pallets with non-identical items," European Journal of Operational Research, vol. 84, pp. 681–692, 1995.

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

[6] H. Gehring and A. Bortfeldt, "A genetic algorithm for solving the container loading problem," International Transactions in Operational Research, pp. 401–418, 1997.

[7] A. Bortfeldt and H. Gehring, “A hybrid genetic algorithm for the container loading problem,” European Journal of Operational Research, vol. 131, pp. 143–161, 2001.

[8] J. F. Gonçalves, M. G. C. Resende, “A parallel multi-population biased random-key genetic algorithm for a container loading problem,” Computer and operations research, vol. 39, pp. 179-190, February, 2012.

[9] D. Mack, A. Bortfeldt and H. Gehring, “A parallel hybrid local search algorithm for the container loading problem,” International Transactions in Operational Research, vol. 11, pp. 511–533, 2004.

[10] 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, 2003.

[11] E. Bischoff, “Three-dimensional packing of items with limited load bearing strength,” European Journal of Operations Research, vol. 168, pp. 952–966, 2004.

[12] A. Moura and J. F. Oliveira, “A grasp approach to the container-loading problem,” IEEE Intelligent Systems, vol. 20, pp. 50–57, 2005.

[13] R. Morabito and M. Arenales, “An AND/OR-graph approach to the container loading problem,” International Transactions in Operational Research, vol. 1, pp. 59–73, 1994.

[14] M. Eley, “Solving container loading problems by block arrangement,” European Journal of Operational Research, vol. 141, pp. 393–409, 2002.

[15] M. Hifi, “Approximate algorithms for the container loading problem,” International Transactions in Operations Research, vol. 9, pp. 747–774, 2002.

[16] D. Pisinger, “Heuristics for the container loading problem,” European Journal of Operational Research, vol. 141, pp. 143–153, 2002.

[17] K. K. Lai and J. W. M. Chan, “Developing a simulated annealing algorithm for the cutting stock problem,” Computers and Industrial Engineering, vol. 32, pp. 115–127, 1997.

[18] V. Cerny, “A thermodynamic approach to the traveling salesman problem: an efficient simulated annealing algorithm,” Journal of optimization theory and applications, vol. 45, January, 1985.

[19] A. Bortfeldt, H. Gehring and Mack D, “A parallel tabu search algorithm for solving the container loading problem,” Parallel Computing, vol. 29, pp. 641–662, 2003.

[20] F. Parreno, R. Alvarez-Valdes, 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, 2008.

[21] F. Parreno, R. Alvarez-Valdes, 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, 2010.