


ITC 4/50 Information Technology and Control Vol. 50 / No. 4 / 2021 pp. 808-826 DOI 10.5755/j01.itc.50.4.29009	A Reduced Variable Neighborhood Search Approach to the Heterogeneous Vector Bin Packing Problem	
	Received 2021/04/27	Accepted after revision 2021/11/25
	 <a href="http://dx.doi.org/10.5755/j01.itc.50.4.29009">http://dx.doi.org/10.5755/j01.itc.50.4.29009</a>	

**HOW TO CITE:** Stakić Đ., Živković, M., Anokić, A. (2021). A Reduced Variable Neighborhood Search Approach to the Heterogeneous Vector Bin Packing Problem. *Information Technology and Control*, 50(4), 808-826. <https://doi.org/10.5755/j01.itc.50.4.29009>

# A Reduced Variable Neighborhood Search Approach to the Heterogeneous Vector Bin Packing Problem

**Đorđe Stakić**

University of Belgrade, Faculty of Economics, Kamenička 6, 11 000 Belgrade, Serbia;  
e-mail: djordje.stakic@ekof.bg.ac.rs

**Miodrag Živković**

University of Belgrade, Faculty of Mathematics, Studentski trg 16, 11 000 Belgrade, Serbia;  
e-mail: ezivkovm@matf.bg.ac.rs

**Ana Anokić**

Academy of Technical and Art Applied Studies Belgrade, School of Applied Studies for Information and Communication Technologies, Zdravka Čelara 16, 11 000 Belgrade, Serbia; e-mail: ana.anokic@ict.edu.rs

**Corresponding author:** djordje.stakic@ekof.bg.ac.rs

The two-dimensional heterogeneous vector bin packing problem (2DHet-VBPP) consists of packing the set of items into the set of various type bins, respecting their two resource limits. The problem is to minimize the total cost of all bins. The problem, known to be NP-hard, can be formulated as a pure integer linear program, but optimal solutions can be obtained by the CPLEX Optimizer engine only for small instances. This paper proposes a metaheuristic approach to the 2DHet-VBPP, based on Reduced variable neighborhood search (RVNS). All RVNS elements are adapted to the considered problem and many procedures are designed to improve efficiency of the method. As the Two-dimensional Homogeneous-VBPP (2DHom-VBPP) is more often treated, we considered also a special version of the RVNS algorithm to solve the 2DHom-VBPP. The results obtained and compared to both CPLEX results and results on benchmark instances from literature, justify the use of the RVNS algorithm to solve large instances of these optimization problems.

**KEYWORDS:** Two-dimensional heterogeneous vector bin packing problem, Variable neighborhood search, Container transport, Optimization, Metaheuristics.

## 1. Introduction

The *two-dimensional heterogeneous vector bin packing problem* (2DHet-VBPP) can be stated as follows (see Han et al. [13], where the name M2BP – multi-type two-dimensional bin packing problem, was used): given the  $N$  pairs (2-dimensional items) and the finite number of bin types, characterized by the capacity and the cost, the problem is to select bins and pack all items into these bins, so that the total cost is minimized and the resource constraints are met.

The problem 2DHet-VBPP is a special case of the *Vector bin packing problem* (VBPP) (see Garey et al. [11]), where bins and items are vectors of dimension  $N$ . The problem *Homogeneous VBPP* (Hom-VBPP) is a special case of the 2DHet-VBPP when there is only one type of bins. The decision variant of the Hom-VBPP is NP-complete (see [12]), hence the VBPP is computationally a hard problem.

Han et al. [13] stated the following problems related to the 2DHet-VBPP:

- assignment of computer processes to processors, taking into account the two resources, processing time and memory;
- assignment of robots to working stations;
- file placement for multi-device storage system.

Gabay and Zaourar [10] considered the Virtual machine placement problem, which is extended by additional constraints into the Machine reassignment problem.

Bin packing is related to the container transport, as well. According to Notteboom [25], every year hundreds of millions of containers have been transported worldwide. Therefore, an efficient optimization of the container transport and packing is very valuable from the global perspectives on environmental concern and minimization of costs. For details about the container transport and related problems, refer to Bortfeldt and Wäscher [5], who are focused on the constraints of packing items into containers. Problems are classified into the minimization and maximization problem types. Among them, the 2DHet-VBPP appears under the name *Multiple Bin-Size Bin Packing Problem* (MBSBPP) and it is described as packing a strongly heterogeneous set of cargo into a weakly heterogeneous assortment of containers so that the price of the used containers is minimized. From the 163 papers considered in [5], 12 of them dealt with

MBSBPP, while 23 papers used weight limits for the container-related constraints and only 2 combined weight limits with some additional constraints in the MBSBPP. Packing the set of items into the set of containers in order to minimize the total cost, taking into account the limits of containers and weights and volumes of items, is considered in [29], [35], [34].

The exact optimization methods cannot solve larger instances of NP-hard problems. Therefore, the heuristic method is a reasonable choice to deal with the 2DHet-VBPP. Han et al. [13] considered the three approaches to the 2DHet-VBPP: a simple greedy heuristic named First Fit by Ordered Deviation (FFOD), Simulated annealing (SA) and Column generation (CG). The fastest method was FFOD, but with the solutions with the largest gap from the best ones; SA was slower, but with a smaller gap. Exact solutions could not be obtained for the instances with more than 50 items. Gabay and Zaourar [10] considered 34 greedy heuristic methods to solve the 2DHet-VBPP, classified into: item centric, bin centric and bin balancing heuristics for the decision variant of the VBPP where the final goal was to answer the question whether the given items could be placed into bins or not. Three metaheuristic methods, based on *Greedy Randomized Adaptive Search Procedure* (GRASP) metaheuristics, are developed by Stakić et al. [34]: Basic GRASP, Uniform GRASP and Reactive GRASP method; the Uniform GRASP method outperformed the remaining two. Stakić et al. [35] used a variant of *Variable neighborhood search* (VNS) to solve the 2DHet-VBPP.

As mentioned earlier, homogeneous variant of the problem was more considered in literature and there are many benchmarks for the 2DHom-VBPP. Approximation algorithms and optimal solutions for some instances of 2DHom-VBPP were represented by Shachnai and Tamir in [32]. In addition, the authors showed that for a single bin the problem was solvable in pseudo-polynomial time. Alves et al. [1] applied the concept of dual-feasible functions to reach fast lower bounds for the 2DHom-VBPP extending 1-dimensional functions to the  $m$ -dimensional case. The authors analyzed different families of functions. A multi-start iterated local search heuristic, relying on simple neighborhoods and problem-tailored shaking procedures, was developed by Masson et al. [22] as a solu-

tion approach for two problems, multi-capacity bin packing and machine reassignment problem. Turkey et al. [36] developed Hyper-heuristic framework based on automatically selecting local search algorithm and the internal operators to solve 2DHom-VBPP denoted as Multi-Capacity Bin Packing Problem (MCBPP). Brunch-and-Price exact algorithms were applied by Heßler et al. [19] to several vector packing problems, among which the 2DHom-VBPP was considered. The authors used the underlying column-generation procedure with the extended master program stabilized by dual-optimal inequalities that were added from the beginning or dynamically. Branch-cut-and-price algorithms, defined as models for VRPSolver (a generic solver for vehicle routing problems) were applied to the classical bin packing problem, as well as vector packing, variable sized bin packing, and variable sized bin packing with optional items by Pessoa et al. [28]. A consistent neighborhood search was developed for solving the one-dimensional bin packing problem and then applied to the 2DHom-VBPP by Buljubašić and Vasquez [7]. Panigrahy et al. [26] analysed variants of the heuristics inspired by the first-fit decreasing algorithm and propose a new geometric heuristic algorithm without a significant decrease in performance. Aringhieri et al. [2] developed the two heuristics: greedy and neighborhood search algorithm for the same problem. Sophisticated branch-cut-and-price algorithms were proposed by Pessoa et al. [27] for solving vehicle routing, assignment and bin packing problems, including the 2DHom-VBPP. In the early work of Spieksma [33] a branch-and-bound algorithm for 2DHom-VBPP was described, but it provided optimal solutions only for small-size instances. However, some instances published in [33] were later used as benchmarks. Wei et al. [37] developed a branch-and-bound method with dynamic programming which, after eliminating conflicts between two items through branching, solves the two-constraint knapsack problem at leaf nodes using dynamic programming. Brandao and Pedroso [6] presented an exact method based on an arc-flow formulation with side constraints for bin packing, including 2DHom-VBPP, and cutting stock problems. In their method, all the patterns formed a very compact graph to which a graph compression algorithm was applied in order to reduce the size of a graph without weakening the model. Each of the 34 greedy heuristics implemented by Gabay and Zaourar [10] were applied

to each fixed set of items and bins of the considered 2DHom-VBPP. Then, a binary search procedure was used to find the minimal number of bins for the considered set of items among all the solutions obtained by these heuristics. Due to the simple implementation of these heuristics, the approach is not time consuming, but it cannot be applied to heterogenous case. Caprara and Toth [8] analyzed several lower bounds and introduced heuristic and exact algorithm for 2DHom-VBPP. Additional information considering different bin packing problems and solution approaches can be found in the work of Christensen et al. [9]. In [9] authors considered Geometric bin packing and Vector bin packing problems. Within the second type they discussed offline and online vector bin packing, Vector knapsack, Vector scheduling and Vector covering problems. The specific 2DHom-VBPP with different categories of products (standard, cooled and frozen) that require separated zones in a truck while avoiding splitting orders with several goals (minimizing the total number of trucks, the number of refrigerated trucks which contain frozen and standard products and minimizing splitting), was considered by Heßler et al. [20]. Another variant of 2DHom-VBPP, where a price of a bin depends on the total mass of items in it, was considered and solved using a memetic algorithm by Hu et al. [21]. 2D-VBPP appeared in a scheduling problem considered by Billaut et al. [4]. Namely, two-dimensional jobs (with duration and consumption) consume a perishable resource stored in vials. The goal is to schedule the jobs on a single machine so that the maximum lateness does not exceed a given threshold and the number of vials required for processing all the jobs is minimized. The two-step approach embedding a Recovering Beam Search algorithm for the initial solution and a metaheuristic algorithm are proposed for the considered problem. The variable neighborhood search algorithm is developed for solving the bin packing problem (BPP) with compatible categories by Santos et al. [31]. A recent survey by Munien and Ezugwu [24] on different solution approaches for the 1DHom-VBPP represents The fitness-dependent optimizer (FDO), Cuckoo search via Lévy flights, Whale optimization algorithm (WOA), Squirrel search algorithm and Genetic algorithms as successful metaheuristic approaches to this type of the Bin packing problem. Another review by Ramos et al. [30] on grouping problems, among which is bin

packing as well, classify the applied metaheuristics as: Neighborhood searches, Evolutionary algorithms, Swarm intelligence algorithms.

Here we present a more efficient metaheuristic approach to the 2DHet-VBPP, based on the Reduced variable neighborhood search (RVNS). The algorithm is tested on both of 2DHet-VBPP and 2DHom-VBPP instances, and the results are compared to [2], [6], [34], [19], [10], [36]. The main contribution of our work is a new solution approach, a variant of a VNS metaheuristics, for a less considered problem, 2DHet-VBPP. Having in mind that exact methods can solve only small-size instances in the case of NP-hard problems and population based metaheuristics, as genetic algorithms, consequently provide a large number of unfeasible solutions, we develop a systematic single solution metaheuristic method. The adaption of our method to the more considered 2DHom-VBPP is successful compared to the benchmarks and provides a several new optimal solutions for the instances from the literature.

The rest of the paper is organized as follows: the proposed RVNS method is described in Section 2, the experimental results are presented in Section 3, Section 4 states the conclusion.

## 2. The Proposed RVNS Algorithm

In order to present the mathematical formulation of the 2DHet-VBPP and introduce the proposed solution method, the precise designations are used. If  $n$  is an integer, let  $[n] = \{1, 2, \dots, n\}$ . Further:

- $(m_i, V_i)$  denotes the two dimensions of item  $i$ ,  $i \in [N]$ .
- $(Lm_t, LV_t)$  denotes the capacity of bins of type  $t$ ,  $t \in [nt]$ .
- $C_t$  denotes the price of bins of type  $t$ ,  $t \in [nt]$ .
- $Ln_t$  denotes the upper bound on the number of bins of type  $t$ ,  $t \in [nt]$ .
- $p_{ijt} = 1$  if item  $i \in [n]$  is packed in the  $j$ th bin ( $j \in [Ln_t]$ ) of type  $t \in [nt]$ ; otherwise  $p_{ijt} = 0$ .
- $k_{jt} = 1$  if the  $j$ th bin,  $j \in [Ln_t]$ , of type  $t \in [nt]$  is used; otherwise  $k_{jt} = 0$ .

The optimization problem 2DHet-VBPP is to

$$\text{minimize } C = \sum_{t=1}^{nt} \sum_{j=1}^{Ln_t} k_{jt} \cdot C_t \quad (1)$$

$$\text{subject to } \sum_{t=1}^{nt} \sum_{j=1}^{Ln_t} p_{ijt} = 1, \quad i \in [n] \quad (2)$$

$$\begin{aligned} \sum_{i=1}^n p_{ijt} \cdot (m_i, V_i) &\leq k_{jt} \cdot (Lm_t, LV_t), \\ t \in [nt], j \in [Ln_t] \\ p_{ijt} &\in \{0, 1\}, \quad i \in [n], t \in [nt], j \in [Ln_t] \\ k_{jt} &\in \{0, 1\}, \quad t \in [nt], j \in [Ln_t] \end{aligned} \quad (3)$$

The objective function  $C$  (1) represents the total costs of used bins. Constraints (2) ensure that each item is packed into exactly one bin. Constraints (3) ensure that bin capacities are not exceeded.

A precise mathematical model can be used by the exact optimization solver. However, it is a well known fact that only small instances of NP-hard optimization problems can be solved exactly. A common approach is to develop an efficient metaheuristic algorithm that will be successful with larger instances of the considered problem.

*Variable neighborhood search* (VNS) is a well-known metaheuristic method, introduced by Hansen and Mladenović [14]-[18], [23] and widely used to solve various continuous and combinatorial optimization problems (as well as BPP by Santos et al. [31]), see Algorithm 1. Starting with the initial solution, the basic VNS algorithm loops repeat the three main steps: *Shaking phase*, *Local search* and *Move or Not* step, until the termination criterion is reached.

### Algorithm 1 basic VNS

```

procedure VNS(Problem Data,  $r_{max}$ )
  Generate initial solution  $S$ ;
  repeat
     $r \leftarrow 1$ ;
    while  $r \leq r_{max}$  do
       $S' \leftarrow Shake(S, r)$ ; //Shaking phase
       $S'' \leftarrow Local Search(S')$ ; //Local search
      if  $f(S'') < f(S)$  then //Move or Not
         $S \leftarrow S''$ ;
         $r \leftarrow 1$ ;
      else
         $r \leftarrow r + 1$ ;
  until The termination criterion is satisfied

```

After generating the initial solution  $S$  (incumbent), the first VNS step (called Shaking phase), directs the search to different points in the search space, enabling the diversification of the search process. The solution  $S'$ , obtained in this phase, is passed to the second step (Local search), the aim of which is to reach the local optimum in the neighborhood of  $S'$ . The best solution  $S''$ , obtained during Local search, is compared to the incumbent  $S$ , and the better one is preserved as new  $S$  by Move or Not VNS step. If  $S''$  is not better than  $S$ , the search continues from  $S$ , and the counter  $r$  of iterations without improvement is incremented. As the termination criterion, there might be used the maximum number of iterations without improvement, the maximum running time, etc. In the presented VNS implementation, a running time limit  $t_{max}$  was used as the termination criterion.

As the neighbourhood of a 2DHet-VBPP feasible solution is very large and complicated, the *Reduced VNS* (RVNS, see Hansen et al. [15], [18]) was used, the variant obtained from the basic VNS by omitting the Local search step.

### 2.1. Solution Representation and Objective Function Calculation

During the 2DHet-VBPP solution process, a sequence of states (feasible solutions) is considered. Let  $n$  denote the number of items. If  $\mathcal{S}$  is a state, then let  $nBin(\mathcal{S})$  denote the number of bins corresponding to  $\mathcal{S}$ , and let  $n^*$  denote some upper bound on the number of items that can be placed into one bin. The state  $\mathcal{S}$  is uniquely represented by a pair  $(S, a)$ , where

- $S$  is an integer  $n \times n^*$  matrix. The row  $j$  of  $S$  corresponds to bin  $j$ . If  $i > 0$ , then  $S[j, k] = i$  means that the item  $i$  is the  $k$ -th item in the bin  $j$ ; otherwise the number of items in the bin  $j$  is less than  $k$ , where  $j, i \in [n], k \in [n^*]$ .
- $a$  is an array of length  $n$ . If  $a[k] > 0$ , then  $a[k]$  is the type of the  $k$  th bin, otherwise  $nBin(\mathcal{S}) < k$ .

Assuming that the weight and volume of each item is less than the corresponding limit of bin, we have  $nBin(\mathcal{S}) \leq n$ . A somewhat better upper bound  $n^*$  is expressed by

$$\max_{1 \leq i \leq n_i} \min \left\{ \max_{1 \leq k \leq n} \sum_{j=1}^k m'_j \leq Lm_i, \max_{1 \leq k \leq n} \sum_{j=1}^k V'_j \leq LV_i \right\}, \quad (4)$$

where  $m'_1 \leq m'_2 \leq \dots \leq m'_n$  is the sorted permutation of  $\{m_1, \dots, m_n\}$  and  $V'_1 \leq V'_2 \leq \dots \leq V'_n$  is the sorted permutation of  $\{V_1, \dots, V_n\}$ . Namely, for each bin type  $t$  we consider the minimum of the two numbers (in the parenthesis) - the maximum number of items that can be placed in the bin of type  $t$  respecting the limit in weight and the limit in volume. The largest of these minimum values over the set of all bin types is  $n^*$ , the maximal number of items that can be placed in bin of each type.

Let  $nBin(S, a) = nBin(\mathcal{S})$ , where  $\mathcal{S}$  is the state corresponding to  $(S, a)$ . From the definition, it follows that the  $i$  th row of  $S$  is all-zero if  $i > nBin(S, a)$ . The objective function value corresponding to  $(S, a)$  is indicated by

$$C = f(S, a) = \sum_{j=1}^{nBin(S, a)} C_{a[j]}.$$

**Example 1.** Consider the instance with  $n = 10$  items and with  $nt = 3$  bin types specified by Tables 1 and 2. We can suppose that volumes are measured in  $m^3$ , that weights are measured in tons, and that the price of using a bin is in euros.

**Table 1**

Example 2DHet-VBPP instance: items

$i$	1	2	3	4	5	6	7	8	9	10
$m_i (t)$	13	5	6	9	9	4	4	10	7	4
$V_i (m^3)$	1	8	23	9	3	21	24	24	17	1

**Table 2**

Example 2DHet-VBPP instance: bins

$i$	1	2	3
$Lm_i (t)$	25.8	24.5	24.5
$LV_i (m^3)$	30	60	70
$C_i (EUR)$	1594	2470	2483

From (4) it follows that the matrix  $S$  corresponding to any feasible solution has at most  $n^* = 5$  columns. The optimal solution  $(S_{opt}, a_{opt})$ , obtained by CPLEX, uses  $nBin(S_{opt}, a_{opt}) = 3$  bins, and it is presented by

$$S_{opt} = \begin{bmatrix} 1 & 9 & 10 & 0 & 0 \\ 2 & 3 & 5 & 6 & 0 \\ 4 & 7 & 8 & 0 & 0 \end{bmatrix}, \quad a_{opt} = [1, 2, 2].$$

The first bin is of type 1, and the remaining two are of type 2. The optimal solution puts, for example, the three items with the indices 1, 9 and 10 into the first bin. Their vectors  $(m_i, V_i)$  are (13,1), (7,17) and (4,1), respectively. The sum of these three vectors is (24,19), which is lexicographically less than the capacity  $(Lm_i, LV_i) = (25.8, 30)$ . The four items 2, 3, 5, 6 are placed in bin 2, and the three remaining items 4, 7, 8 are placed in the bin 3. The objective function value for the optimal solution is hence  $C_{opt} = f(S_{opt}, a_{opt}) = 1594 + 2 \cdot 2470 = 6534$  EUR.

### 2.2. Generating the Initial Solution

The initial solution is generated using the simple greedy algorithm, similar to decreasing-first-fit heuristic for the 1D bin packing problem. The type of all bins in the initial solution is set to 1. The pairs  $(m_i, V_i)$  are sorted lexicographically in a decreasing order. Following this order, each item is placed into the first (from the beginning) bin in which it fits. If it is not possible to place the current item into any of the already used bins, then it is placed in a new, empty bin.

**Example 2.** Consider the instance from *Example 1*. The indices of the lexicographically sorted list of items are (1, 8, 4, 5, 9, 3, 2, 7, 6, 10). The order in which items are placed into 5 bins by the first-fit rule is shown in Table 3.

**Table 3**  
Example 2DHet-VBPP instance: generating initial solution

$i$	1	8	4	5	9	3	2	7	6	10
$m_i$	13	10	9	9	7	6	5	4	4	4
$V_i$	1	24	9	3	17	23	8	24	21	1
bin index(j)	1	1	2	2	2	3	4	5	4	3
sum $m_i$ for bin j	13	23	9	18	25	6	5	4	9	10
sum $V_i$ for bin j	1	25	9	12	29	23	8	24	29	24

The fourth row indicates the chosen bin index, and the corresponding entries in the next two rows show the sums of the first and the second coordinates of

items placed into this bin until this step. Hence, the components of the initial solution  $(S_{init}, a_{init})$  are:

$$S_{init} = \begin{bmatrix} 1 & 8 & 0 \\ 4 & 5 & 9 \\ 3 & 10 & 0 \\ 2 & 6 & 0 \\ 7 & 0 & 0 \end{bmatrix}, \quad a_{init} = [1, 1, 1, 1, 1].$$

The objective function value  $C_{init}$  corresponding to the initial solution  $(S_{init}, a_{init})$  is  $C_{init} = f(S_{init}, a_{init}) = 5 \cdot 1594 = 7970$  EUR, which is 1436 EUR more than the corresponding optimal value shown in Example 1.

### 2.3. The RVNS Algorithm

What follows is the list of certain terms and their designations, as well as a description of a number of simple procedures.

- Let  $random(a, b)$  denotes randomly generated integer value from the interval  $[a, b]$ .
- Let  $nBin(S)$  denotes the number of bins in the solution corresponding to the matrix  $S$ .
- Let  $nItem(S, j)$  denotes the number of items in the bin  $j$ , which is the part of the solution corresponding to the matrix  $S$ .
- Let  $load(S, j) = \sum_{i=1}^n p_{ija_j} \cdot (m_i, V_i)$  denotes the total load of the bin  $j$ , which is the part of the solution corresponding to the matrix  $S$ .
- Let  $f(S, a)$  denotes the value of the objective function.
- Let  $nFBin(S, a)$  denotes the the number of full bins, i.e. the number of bins  $j$ , such that  $load(S, j) = (Lm_{a_j}, LV_{a_j})$ .
- Let  $Swap(S, j_1, i_1, j_2, i_2)$  denotes the procedure (see Algorithm 2)

If  $j_1, j_2 > 0$ , then

**If**  $i_1 > 0$  **and**  $i_2 > 0$ , then swap (if possible) the item  $i_1$  from the bin  $j_1$  and the item  $i_2$  from the bin  $j_2$ ;

**If**  $i_1 = 0$  **and**  $i_2 > 0$ , then transfer (if possible) the item  $i_2$  from the bin  $j_2$  into the bin  $j_1$ ;

**If**  $i_2 = 0$  **and**  $i_1 > 0$ , then transfer (if possible) the item  $i_1$  from the bin  $j_1$  into the bin  $j_2$ ;

**If**  $i_1 = i_2 = 0$ , then do nothing;

---

**Algorithm 2** The proposed RVNS algorithm
 

---

```

procedure RVNS(Problem Data,  $r_{max}$ ,  $t_{max}$ )
  Generate initial solution ( $S, a$ );
  repeat
     $r \leftarrow 1$ ;
    while  $r \leq r_{max}$  do
      ( $S', a'$ )  $\leftarrow$  ( $S, a$ );           //Shaking phase
      for ( $k \leftarrow 1; k \leq r; k++$ ) do
        //randomly change type of randomly chosen bin
        ( $j, t$ )  $\leftarrow$  ( $random(1, nBin(S')), random(1, nt)$ );
        if ( $load(S', j) \leq capacity(t)$ ) then  $a'(j) \leftarrow t$ ;
        // swap two randomly chosen items from two randomly chosen bins
        ( $j_1, j_2$ )  $\leftarrow$  ( $random(0, nBin(S')), random(0, nBin(S'))$ );
        ( $i_1, i_2$ )  $\leftarrow$  ( $random(0, nItem(S', j_1)), random(0, nItem(S', j_2))$ );
         $S' \leftarrow Swap(S', j_1, i_1, j_2, i_2)$ ;
      for ( $k \leftarrow 1; k \leq r; k++$ ) do
         $j \leftarrow random(1, nBin(S'))$ ; // attempt to empty the randomly chosen bin
         $S' \leftarrow EmptyAMAP(S', j)$ ;
      if ( $S', a$ ) is not changed then
         $r \leftarrow r + 1$ ;
        continue;
      ( $S'', a''$ )  $\leftarrow sorted(S', a')$ ;           //Improvement
      for ( $iter \leftarrow 1; iter \leq niter; iter++$ ) do
        for ( $j \leftarrow 1; j \leq nBin(S''); j++$ ) do
          for ( $i \leftarrow nItem(S'', j); i \geq 1; i--$ ) do
            for ( $k \leftarrow nBin(S''); k \geq 1; k--$ ) do
              if ( $TransferOrSwap(S'', j, i, k)$ ) then
                break;
            if ( $EmptyBin(S'', j)$ ) then
              ( $S'', a''$ )  $\leftarrow sorted(S'', a'')$ ;
      ( $S'', a''$ )  $\leftarrow ImproveByType(S'', a'')$ ;
      if  $f(S'', a'') < f(S, a)$  then           //Move or Not
        ( $S, a$ )  $\leftarrow$  ( $S'', a''$ );
         $r \leftarrow 1$ ;
      else
        if ( $f(S'', a'') = f(S, a)$  and  $nFBin(S'') \geq nFBin(S)$ ) then
          ( $S, a$ )  $\leftarrow sorted(S'', a'')$ ;
         $r \leftarrow r + 1$ ;
  until  $SessionTime \geq t_{max}$ 

```

---

If  $j_1 = 0, j_2 > 0$ , and  $i_2 > 0$ , then increment  $nBin(S)$ , and transfer the item  $i_2$  from the bin  $j_2$  into the new empty bin;

If  $j_2 = 0, j_1 > 0$ , and  $i_1 > 0$ , then increment  $nBin(S)$ , and transfer the item  $i_1$  from the bin  $j_1$  into the new empty bin;

If  $j_1 = j_2 = 0$ , then do nothing.

- Let  $EmptyAMAP(S, j)$  denotes the transfer (as much as possible) of items from the  $j$ th bin,  $j \in [nBin(S)]$ , into all other bins  $i \in [nBin(S)] \setminus \{j\}$  starting from the last one. After  $EmptyAMAP(S, j)$  is applied, the bin  $j$  might be emptied.
- Let  $sorted(S, a)$  denotes the procedure which sorts the items in each bin of the solution in increasing order according to their volume, as well as the set of bins according to the total volume of their load.
- Let  $TransferOrSwap(S, j, i, k)$  denotes the procedure that
  - if the item  $i$  from the bin  $j$  fits into bin  $k$ , then move it from the bin  $j$  into bin  $k$ ;
  - otherwise, for all the items  $l, l \in [nItem(S, k)]$  in the bin  $k$ , try to swap the item  $i$  from the bin  $j$  and the item  $l$  from the bin  $k$  until the first such swap occurs; in that case it return *True*; otherwise, return *False*. More precisely, in order to empty (if possible) the considered bin, it is allowed to replace an item only by the one with a smaller volume. As in the shaking phase, this move is also performed if the bin can be partially emptied.

For the specification of the proposed RVNS algorithm, see Algorithm 2. The unsuccessful shakings counter  $r$  is initialized to 1. The body of the main loop is repeated until the counter  $r$  reaches the limit  $r_{max}$ . The counter is incremented if the execution of the body leads to no improvement; otherwise,  $r$  is reset to 1. Passing the  $r$  loop is repeated while the limit of the session time  $t_{max}$  is reached. The body of the main loop consists of:

- the Shaking phase
- the Improvement step
- the Move or Not step

The Shaking phase of our RVNS algorithm starts from feasible solution  $(S', a')$  obtained as a copy of the current best solution  $(S, a)$ . The Shaking phase consists

of the following moves, as described in Algorithm 2.

- 1 Repeat  $r$  times the following two steps
  - a Change the type of one randomly selected bin  $j \in [nBin(S')]$  to randomly selected type  $t$ , if possible, i.e. if  $load(S', j) \leq (Lm, LV_t)$  (in other words, set  $a'[j] \leftarrow t$ ); otherwise do nothing.
  - b Choose a random pair of different bins  $j_1, j_2 \in \{0\} \cup [nBin(S')]$  (independently, from the uniform distributions). Choose a random pair of items  $i_1 \in \{0\} \cup [nItem(S', j_1)]$  and  $i_2 \in \{0\} \cup [nItem(S', j_2)]$  (independently, from the uniform distributions). Perform the procedure  $Swap(S', j_1, i_1, j_2, i_2)$  in order to change the solution  $(S', a')$  (the probability to choose  $i_1 = 0$ , for example, equals to  $1 / (nItem(S', j_1) + 1)$ ).
- 2 Execute the procedure  $EmptyAMAP(S', j)$   $r$  times, for  $r$  randomly chosen bins  $j$ .

In case the solution  $(S', a')$  is not changed after the Shaking phase, the Shaking phase is repeated, with  $r$  incremented. Otherwise, the Improvement step follows.

The Improvement step is the replacement for the Local search phase in VNS algorithm. This step is a reduced version of the local search, because the complete local search here is too complex. One can alternatively consider the Improvement step as the final part of the shaking phase. The sorted copy  $sorted(S', a')$  of  $(S', a')$  is saved as  $(S'', a'')$ . Starting from the first bin (the least loaded one) in  $(S'', a'')$ ,  $niter$  iterations of the triple for loop by  $j, i$  and  $k$  with the body  $TransferOrSwap(S'', j, i, k)$  are performed in order to reduce the number of bins by moving the items, the largest first, to the more loaded bins. In this process, completely full bins are skipped. After exiting the  $j$  loop, it is checked if the bin  $j$  is empty (it is possible that the bin  $j$  is not empty); if it is empty, then  $(S'', a'')$  is replaced by  $sorted(S'', a'')$ .

After the Improvement step, all the bins are checked, and if possible, replaced by a smaller cost bin type ( $ImproveByType(S'', a'')$  step), after which the Move or Not step follows:

- If  $f(S'', a'') < f(S, a)$ , then  $(S, a)$  is replaced by  $(S'', a'')$ , and the counter  $r$  is reset;
- Otherwise, if  $f(S'', a'') = f(S, a)$  and  $nFBin(S'', a'') > nFBin(S, a)$ , then  $(S, a)$  is replaced by  $(S'', a'')$ , and the counter  $r$  is incremented. This small



modification of the classic Move or Not phase in VNS method performs better, as it enables a diversification of the search, by starting from the different solution even if it has the same objective function value.

- otherwise,  $r$  is incremented.

When the time limit  $t_{max}$  is reached, our RVNS algorithm stops, returning the best solution found. It is a straightforward task to generalize this algorithm to the case of arbitrary dimension.

#### 2.4. The Adaptation of the Proposed RVNS Methods for Hom-VBPP

Regarding the fact that numerous papers have been published studying the 2DHom-VBPP, the RVNS algorithm is also adapted to solve the 2DHom-VBPP. In fact, the 2DHom-VBPP is a special case of the 2DHet-VBPP, where all the bins are of the same type, i.e.  $nt = 1$ . Hence, in the shaking step the first move (changing the type of a randomly selected bin) is skipped. As the objective function value is proportional to the number of used bins, it can be replaced by the number of bins used, see (1):

$$C' = \sum_{j=1}^{Ln_1} k_{j1}. \quad (5)$$

There is a simple lower bound for this objective function considered by Gabay and Zaourar [10]:

$$l_{\infty} = \max \left\{ \left\lceil \frac{\sum m_i}{Lm} \right\rceil, \left\lceil \frac{\sum V_i}{LV} \right\rceil \right\}. \quad (6)$$

If  $C' = l_{\infty}$ , then the objective function value  $C'$  is optimal. Following [10], we used this fact in some cases to prove the optimality of the objective function value.

### 3. The Experimental Results

The newly generated 2DHet-VBPP instances and the 2CBP set of 2DHom-VBPP benchmark instances provided by Caprara and Toth [8] are used to evaluate the RVNS algorithm. The proposed RVNS algorithm is implemented in C programming language. All experimental results were obtained using Intel Xeon CPU

E5-2620 v3, 2.40 GHz with 32GB RAM memory, under Linux operating system. CPLEX 12.6.2 solver was used to solve some smaller instances. Before proceeding to the evaluation, we performed tests in order to adequately choose the value of the parameters  $niter$  and  $r_{max}$ .

#### 3.1. 2DHet-VBPP Instances

The ensuing part of the paper describes the benchmark 2DHet-VBPP used to evaluate the RVNS algorithm.

In fact, to the best of our knowledge, there are no other published benchmark instances for the 2DHet-VBPP. Instead, the methods used to generate instances are described.

- Han et al. [13] used a data set of 550 instances. They involved 4 types of bins with different costs: 10, 8, 7, and 6 \$, while the considered numbers of items were: 30, 40, 50, 75, and 100. The average item sizes of instances were 5%, 10% or 20% of the corresponding bin capacity. The update rate of item's size (an item can slightly increase its size) were chosen to be 1%, 2%, 3%, 4% or 5%. Out of 75 different combinations, the authors selected 55, and generated 10 instances for each combination, creating the data set of 550 instances. The first and the second item dimensions were generated following the uniform distribution, and the exponential distribution, respectively.
- Gabay and Zaourar [10] generated the total of 4500 instances, the 100 instances in each combination of:
  - the 5 classes with different input data (Random uniform, Random uniform with rare resources, Correlated capacities, Correlated capacities and requirements and Similar items and bins);
  - the number of bins (10, 30 or 100);
  - the number of dimensions (2, 5 or 10).

The values for both dimensions of items were generated following a uniform distribution with limits depending on the class of data. The 2DHet-VBPP problem on these instances was treated as a decision problem - whether the items could be placed in the given bins set, or not.

- The two data sets, the set of 6 small and the 7 large instances were considered by Stakić et al. [35], and the set of large instances was extended

with 3 more instances by the same first author [34]. The data related to the items were uniformly generated at random within the specific range, while the data related to bins were real-life values corresponding to the properties of the containers that were involved, see Table 2.

- The 10 large instances were considered in [34].

What follows is a description of our newly generated 2DHet-VBPP instances. The bin types were fixed – they corresponded to three container types: 20' container, 40' container and 40' higher container, and their costs were the prices of transport from Shanghai to Belgrade by Rajković et al. [29], see Table 2. The same type of bins were used in [34] and [35]. Six small instances with 10, 11, 12, 13, 15 and 20 items, as well as 50 randomly generated large instances were considered. Weights and volumes of items were randomly uniformly chosen integer values from [1,15] tons and [1,25]m<sup>3</sup>, respectively. The set of large instances had 5 instances with each of the following numbers of items: 50, 70, 100, 120, 150, 200, 350, 500, 750 and 1000.<sup>1</sup> We assumed that the number of available bins for each type was infinite, i.e. it was equal to the number of items.

### 3.2. Choosing the Parameter Values

Our RVNS algorithm depends on the following three parameters:

- $r_{max}$  - the upper bound on the number of shaking steps without any improvement,
- $niter$  - the number of iterations in improvement phase,
- $t_{max}$  - the maximal running time.

The parameter values are experimentally determined for  $r_{max}$  and  $niter$ . For the parameter tuning tests, we selected a subset of 10 generated large instances that included one instance for each considered number of items.

First, different values for  $r_{max}$  are considered. Let  $\lfloor x \rfloor$  denote  $x$  rounded down to the nearest integer. We decided to try with linear functions  $r_{max} = \lfloor 0.05kn \rfloor$ ,  $1 \leq k \leq 10$ . In our preliminary tests, formula  $r_{max} = \lfloor 0.25n \rfloor$  showed the best performance, so we decided to use this formula and test the 10 values for parameter  $niter \in \{1, \dots, 10\}$ . For each such value, the RVNS algorithm was run 10 times on each instance, with the running time limit of 20 s. The parameter tuning tests for  $niter$  are shown in Table 4, where

**Table 4**  
Parameter tuning tests for  $niter$

$n$ $niter$	50	70	100	120	150	200	350	500	750	1000
1	<b>31517</b>	47094	64291	<b>65906</b>	86423	120226	209653	296247	444569	596963
2	<b>31517</b>	47094	64291	<b>65906</b>	<b>85586</b>	119402	207966	294560	443719	595395
3	<b>31517</b>	47094	64291	<b>65906</b>	<b>85586</b>	<b>119389</b>	208803	294560	442151	594545
4	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	293723	<b>440477</b>	593721
5	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	293723	<b>440477</b>	593708
6	<b>31517</b>	<b>46257</b>	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	<b>292886</b>	<b>440477</b>	<b>592871</b>
7	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	<b>292886</b>	<b>440477</b>	592884
8	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	293723	<b>440477</b>	<b>592871</b>
9	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	293723	<b>440477</b>	<b>592871</b>
10	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	<b>292886</b>	<b>440477</b>	<b>592871</b>

<sup>1</sup> for the set of instances see <https://doi.org/10.5281/zenodo.5319708>

**Table 5**Parameter tuning tests for  $r_{max}$ 

$r_{max} \backslash n$	50	70	100	120	150	200	350	500	750	1000
$\lfloor 0.05n \rfloor$	32354	47931	64291	66743	86423	121063	209640	295397	442151	596219
$\lfloor 0.10n \rfloor$	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	86423	120226	208803	<b>292886</b>	441314	593708
$\lfloor 0.15n \rfloor$	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	86423	120226	208803	293723	441314	593708
$\lfloor 0.20n \rfloor$	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	293723	<b>440477</b>	593708
$\lfloor 0.25n \rfloor$	<b>31517</b>	<b>46257</b>	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	<b>292886</b>	<b>440477</b>	<b>592871</b>
$\lfloor 0.30n \rfloor$	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	293723	<b>440477</b>	<b>592871</b>
$\lfloor 0.35n \rfloor$	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	<b>207248</b>	293723	<b>440477</b>	<b>592871</b>
$\lfloor 0.40n \rfloor$	<b>31517</b>	47107	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	207979	<b>292886</b>	<b>440477</b>	<b>592871</b>
$\lfloor 0.45n \rfloor$	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	208803	293723	<b>440477</b>	593708
$\lfloor 0.50n \rfloor$	<b>31517</b>	47094	<b>63454</b>	<b>65906</b>	<b>85586</b>	<b>119389</b>	207966	293723	<b>440477</b>	<b>592871</b>

the best objective function value is bolded for each instance. As it can be seen from Table 4, the value for parameter  $niter = 6$  provided the best solutions on all 10 tested instances.

Using this value  $niter = 6$  we tested the described formulae for  $r_{max}$  on the same set of instances and with the same time limit. The results are shown in Table 5. Having in mind that the expression  $r_{max} = \lfloor 0.25n \rfloor$  offers the largest number of best solutions on this set of instances, we have confirmed and fixed  $r_{max} = \lfloor 0.25n \rfloor$  as the parameter value in the RVNS algorithm.

When it comes to the parameter  $t_{max}$ , the maximal running time, we used standard values 1s, 10s, 60s, 360s and 1440s, which provide a fair comparison with the results from literature.

### 3.3. 2DHet-VBPP

In order to compare different approaches to the 2DHet-VBPP, we treated the 6 small and the 50 large instances (see section 3.1)

- by the exact solver CPLEX v. 12.6.2. applied to the 2DHet-VBPP PILP model (see Section 1), with the time limit set to 1800s,
- by the RVNS algorithm, and

- by a slightly modified version of the U-GRASP algorithm (Greedy Randomized Adaptive Search Procedure, see Stakić et al. [34]).

The time limit to the RVNS and U-GRASP was set to 1s for small instances, and to 60s for large instances. Both these algorithms were repeated 30 times for each instance, and the best obtained result was registered. In order to estimate the stability of these two algorithms, for each instance the average percentage gap

$$\frac{1}{30} \sum_{i=1}^{30} C_i - \min_{1 \leq i \leq 30} C_i}{\min_{1 \leq i \leq 30} C_i} \times 100\%$$

of the solutions  $C_1, C_2, \dots, C_{30}$  obtained in 30 runs was also registered.

Table 6 shows the optimal solutions and the corresponding CPLEX running times for small instances with  $n \leq 20$ ; the instances are identified by the corresponding number  $n$  of items. Our RVNS and U-GRASP implementations reached all these optimal solutions in much shorter time in all the 30 runs for each instance.

**Table 6**  
Results on small instances

<i>Instance (n)</i>	<i>opt. sol.</i>	<i>Time(s)</i>
10	6534	0.10
11	7252	0.06
12	8846	0.06
13	9722	0.15
15	12047	0.35
20	13786	6.15

Table 7 shows the results obtained for large instances with  $n \geq 50$ . The  $i$ th instance with  $n$  items is de-

noted by  $n_i, 1 \leq i \leq 5$ . For each instance, the CPLEX solution (with the time limit set to 1800s) and the corresponding lower bound were obtained. The CPLEX solution was improved by using the same initial solution as for the RVNS, see section 2.2. Furthermore, for

- RVNS with the time limit set to 1s,
- RVNS with the time limit set to 60s, and
- U-GRASP with the time limit set to 60s,

the best of 30 solutions and the corresponding gap are listed. It is worth mentioning that with 30 consecutive runs the total RVNS running time was 30 s and 1800s for each instance, respectively. The best objective function value among the obtained solutions is shown in bold.

**Table 7**  
Comparison of results on large instances

<i>Instance</i>	<b>CPLEX 1800s</b>		<b>RVNS 1s</b>		<b>RVNS 60s</b>		<b>U-GRASP 60s</b>	
	<i>best</i>	<i>LB</i>	<i>best</i>	<i>gap(%)</i>	<i>best</i>	<i>gap(%)</i>	<i>best</i>	<i>gap(%)</i>
50_1	31530	30398.95	<b>31517</b>	1.7	<b>31517</b>	0	31543	2.59
50_2	31491	29368.89	<b>30615</b>	0.3	<b>30615</b>	0	31359	0.45
50_3	32577	30721.37	<b>31009</b>	5.03	<b>31009</b>	0.99	32577	0.03
50_4	27624	26393.31	<b>26761</b>	2.5	<b>26761</b>	0	27611	0.15
50_5	<b>28500</b>	27382.4	<b>28500</b>	0	<b>28500</b>	0	28513	0.04
70_1	47996	45349.16	47107	1.9	<b>46257</b>	2.06	47996	1.42
70_2	44624	42550	<b>43761</b>	2.36	<b>43761</b>	0.57	45487	0.21
70_3	43564	41353.78	<b>42675</b>	1.57	<b>42675</b>	0	43551	1.67
70_4	43577	40751.86	<b>41957</b>	1.49	<b>41957</b>	0	43722	1
70_5	43932	41871.11	<b>43030</b>	1.05	<b>43030</b>	0	44637	0.38
100_1	65048	62834.24	64291	0.83	<b>63454</b>	0.7	66787	1.05
100_2	61453	58350.92	<b>58996</b>	1.61	<b>58996</b>	0	62329	1.04
100_3	58812	55537.24	<b>56329</b>	1.75	<b>56329</b>	0	59675	1.13
100_4	64654	61209.88	62184	2.16	<b>62171</b>	0.85	64641	2.01
100_5	64444	60751.35	62666	0.97	<b>61961</b>	0.46	66196	1.56
120_1	69252	65301.77	<b>65906</b>	1.66	<b>65906</b>	0	70102	1.73
120_2	79355	74939.4	76859	1.27	<b>76009</b>	1.14	81055	0.87
120_3	76206	72707.13	74560	0.94	<b>73723</b>	0.76	77004	1.02
120_4	72269	69002.17	69799	2.15	<b>69773</b>	0.64	75747	1.05
120_5	73526	70048.06	<b>71043</b>	1.66	<b>71043</b>	0.12	75278	1.2
150_1	88958	84667.92	86423	0.81	<b>85586</b>	0.16	90606	1.76

Table 7 (continued)

Instance	CPLEX 1800s		RVNS 1s		RVNS 60s		U-GRASP 60s	
	<i>best</i>	<i>LB</i>	<i>best</i>	<i>gap(%)</i>	<i>best</i>	<i>gap(%)</i>	<i>best</i>	<i>gap(%)</i>
150_2	94787	91176.05	<b>92291</b>	1	<b>92291</b>	0	97205	1.57
150_3	88437	84982.07	86778	0.26	<b>85941</b>	0	90150	1.14
150_4	93911	89060.07	90684	1.24	<b>89847</b>	0.56	95624	0.76
150_5	89650	85053.45	<b>86436</b>	0.74	<b>86436</b>	0	91547	0.96
200_1	124394	118014.1	120239	0.59	<b>119389</b>	0.12	127965	1.69
200_2	123137	116939.4	118995	0.5	<b>118158</b>	0.05	128302	1.53
200_3	126496	121239.4	123230	1.2	<b>122393</b>	0.3	133294	1.48
200_4	119428	111622.5	113863	0.25	<b>113026</b>	0	120674	0.85
200_5	125436	119045.8	121281	0.47	<b>120431</b>	0.14	130759	1.44
350_1	225215	206110.5	207979	1.11	<b>206411</b>	0.45	225810	1.15
350_2	227514	210898.4	213940	0.69	<b>212253</b>	0.12	230856	0.79
350_3	217797	201371.7	202168	1.41	<b>202155</b>	0.01	218965	0.77
350_4	219702	206178.2	208308	0.62	<b>207471</b>	0	225105	0.87
350_5	219023	207275.7	209197	1.22	<b>208347</b>	0.01	226621	0.67
500_1	318066	291891.1	294560	0.56	<b>292886</b>	0.14	322001	0.67
500_2	339309	294525	296364	0.67	<b>295527</b>	0.01	323463	1.05
500_3	361838	284648.9	286616	0.81	<b>284929</b>	0.25	312945	1.07
500_4	382560	308367.5	312765	0.71	<b>310215</b>	0.26	338112	0.91
500_5	319323	290365.4	293150	0.53	<b>291476</b>	0.08	318523	0.68
750_1	546742	439092.3	443825	0.42	<b>439640</b>	0.18	482636	1.02
750_2	537178	442327.9	445842	0.52	<b>443318</b>	0.18	487296	1.55
750_3	554712	443803.6	447467	0.47	<b>445062</b>	0.05	491759	0.77
750_4	548336	442099.5	447184	0.6	<b>442986</b>	0.13	487524	0.83
750_5	551524	448636.8	452394	0.67	<b>449857</b>	0.07	494006	1
1000_1	739616	0	597906	0.46	<b>592871</b>	0.05	655885	0.76
1000_2	733240	0	589508	0.55	<b>582799</b>	0.14	643442	0.67
1000_3	750774	0	606806	0.51	<b>598529</b>	0.42	664521	1.06
1000_4	728458	0	601589	0.35	<b>594999</b>	0.18	657215	1.05
1000_5	744398	0	606640	1.23	<b>599094</b>	0.07	661608	0.63

As expected, the CPLEX algorithm could not find the exact solutions within the time constraint. Table 7 shows that for all these instances the RVNS algorithm with the time limit of 60 s reached the best solutions. The average percentage gap was less than 1% and 0.5%, when the time limit was 1s and 60 s,

respectively, for most of the instances. The U-GRASP algorithm with the time limit of 60 s provided stable, but lower quality solutions for all these instances.

In an additional experiment, we tried to solve these large instances using the CPLEX solver, taking as the initial solution the best RVNS solution. With the

running time extended to 1 h and using all 8 cores, not even one solution was improved. However, CPLEX proved that the obtained RVNS solution for the instance 50\_5 was optimal.

Having in mind the small difference between the lower bound and objective function value of RVNS solutions, an additional effort was made to prove the optimality of some solutions from Table 7. Taking into account the costs of using the considered three types of bins (1594, 2470, 2483), the total cost function can be expressed as:  $C = 1594a + 2470b + 2483c$ , where  $a$ ,  $b$  and  $c$  denote the number of bins for each of the three types. Let  $LB$  denotes the lower bound of the objective function value obtained by the CPLEX and  $RVNS$  denotes the objective function value of the best RVNS solution. For each instance, we generated the set of triples  $(a, b, c)$  that satisfied the relation  $LB \leq C \leq RVNS$  with the additional constraint ensuring that the total capacity of bins from  $(a, b, c)$  was enough for packing all items of the considered instance in terms of weight and volume, i.e.  $\sum m_i \leq 25.8a + 24.5b + 24.5c$  and  $\sum V_i \leq 30a + 60b + 70c$ . If there was exactly one triple  $(a, b, c)$  that fulfilled these conditions, the corresponding value  $C = RVNS$  would be the objective function value of the optimal solution. Using this method, the optimality of the solutions for the following instances was proved: 50\_3, 50\_4, 100\_1, 100\_2, 120\_1 and 350\_1, which resulted in 7 optimal solutions, including the instance 50\_5, provided by our RVNS.

With an additional extension of the CPLEX running time from 1 h to 5 h and using all 8 cores, there were no changes in LB or obtained CPLEX solutions for all instances, except with 1000 items, where the LB values were improved to 591583.60 for 1000\_1, 581894.37 for 1000\_2, 597452.82 for 1000\_3, 594256.58 for 1000\_4 and 597665.20 in the case of the instance 1000\_5. These lower bounds are very close to the objective function value of our RVNS solutions.

As the CPLEX couldn't prove the optimality or obtain better solutions in comparison to our RVNS, we extended significantly the running time for RVNS to 360 s for each run, total 3 h for 30 consecutive runs. We obtained better solution for the following 4 instances: 100\_5 (61124), 500\_4 (309378), 750\_3 (444225) and 750\_5 (449020). An additional extension of running time per each RVNS run from 360 s to 1440 s leads to the improvement of the solution for the instance 1000\_3 (597692). Finally, for these 5 in-

stances, we ran the CPLEX again, starting with these new best found solutions. However, for 5 h of running time and using all 8 cores, these solutions and the corresponding lower bounds were not improved. The optimality of the obtained solutions was not proved for those 5 instances.

### 3.4. 2DHom-VBPP

There are several published sets of benchmark instances for 2DHom-VBPP. We used the data set 2CBP of 400 instances, see Caprara and Toth [8]. These instances are divided in classes (the first three classes were introduced by Spieksma [33]); each class consists of 4 groups of 10 instances with the same number of items. For example, in classes denoted by 1, 2, ..., 9, these 4 groups include instances with 25, 50, 100, and 200 items, respectively. The exception is the last class, denoted by 10, which has 4 groups of instances, with 24, 51, 99 and 201 items, respectively. An instance is *solved* if its optimal solution is found, i.e. if the solution is equal to the corresponding lower bound (6) (see Caprara and Toth [8]). The benchmark authors solved 212 instances. In the work of Brandao and Pedroso [6] the 330 instances were solved<sup>2</sup>. Out of the 70 unsolved instances, the 52 instances were solved by Gabay and Zaourar [10]. Having in mind the total number of solved instances from the set 2CBP, the best results were achieved by Heßler et al. [19] where 370 of 400 instances were solved; the unsolved 30 instances include 200 items and belong to classes: 1, 4, 5, 9, and 10. On the other hand, Pessoa et al. [27] solved 35 of 40 instances from this set, considering only the first four classes: 1, 4, 5 and 9. In the work of Aringhieri et al. [2] and Turkey et al. [36], authors treated all 200 instances from the following 5 classes: 1, 6, 7, 9, and 10, solving 133 and 119 instances, respectively.

With the four different time limits 1 s, 10 s, 60 s and 360 s for a single RVNS run, we tested our adapted RVNS algorithm (Section 2) by running it 30 times for each 2CBP instance. Let  $o(i, j)$  denote the number of solved instances in the group  $j$  of the class  $i$ , if the time limit for a single RVNS run was set to 360 s,  $(i, j) \in [10] \times [4]$ . Under the four different time constraints, the total number  $\sum_{i=1}^{10} \sum_{j=1}^4 o(i, j)$  of in-

<sup>2</sup> <https://research.fdabrandao.pt/research/vpsolver/results/2cbp.html#2cbp>

**Table 8**

The number of Hom-VBPP instances from *2CBP* solved to optimality by the adapted RVNS method with the imposed time limit of 360 s, compared to that of [6]

$j$ : $n$ : $i$	1 (25,24)		2 (50,51)		3 (100,99)		4 (200,201)	
	[6]	RVNS	[6]	RVNS	[6]	RVNS	[6]	RVNS
	1	10	10	10	10	10	6 #	10
2	10	10	10	10	10	10	10	10
3	10	10	10	10	10	10	10	10
4	10	10	0	10 *	0	10 *	0	9 *
5	10	10	0	10 *	0	10 *	0	10 *
6	10	10	10	10	10	3 #	10	0 #
7	10	10	10	10	10	7 #	10	9 #
8	10	10	10	10	10	10	10	10
9	10	10	10	10	10	10	0	0 +
10	10	10	10	10	10	10	10	9

stances solved by RVNS was 315,331,345 and 356, respectively. Out of the 70 unsolved instances by Brandao and Pedroso [6], the 59 instances were solved by RVNS with the time limit of 360 s for each RVNS run. They belong to 6 groups  $(i, j) \in [10] \times [4]$  marked with "\*" in Table 8. Gabay and Zaourar [10] solved 52 of these 70 instances, which means that among these 59 optimal solutions obtained by our RVNS, 7 of them were new.

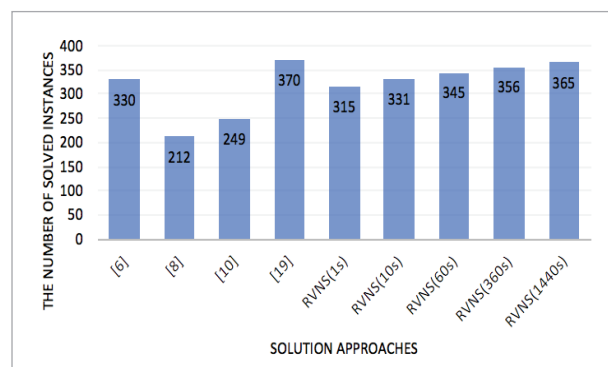
Table 8 lists the numbers  $o(i, j)$ , accompanied with the number of solved instances in [6],  $(i, j) \in [10] \times [4]$ . The number of items corresponding to class  $i$ , group  $j$  equals the first (second) component of the pair for classes 1–9 (for the class 10),  $(i, j) \in [10] \times [4]$ .

Some instances solved in [6] were not solved by RVNS; they are marked by "#". With the running time limit extended from 360 s to 1440 s, the number of solved instances increased from 356 to 365 (the 6 new instances in the group (1,4), one new instance in the group (4,4) and two in (6,3)). Namely, of 70 unsolved instances, our RVNS solved 59 for the time limit of 360 s and another one for 1440 s. Finally, there are instances, marked by "+" in Table 8, that were not solved in the group (9,4), either in [6], or by the RVNS. These instances are regarded as very difficult and Wei et al. [37] were the first who succeeded in solving 1 out of 10 instances in this group.

The main contribution of this research is finding of 59 and 7 new optimal solutions among the 70 instances from the data set *2CBP*, which were not solved in [6] and [10], respectively. Considering the entire set *2CBP* of 400 instances, our RVNS solved 356 instances, while in [6] and [10], the total number of solved instances was 330 and 249, respectively. In [6], the imposed time limit was 12 h for each instance, although the optimal solutions were obtained in shorter running time. The extension of the time limit in our RVNS to 12 h resulted in the total of 365 optimal solutions on the considered set of 400 instances. These results are displayed in Figure 1.

**Figure 1**

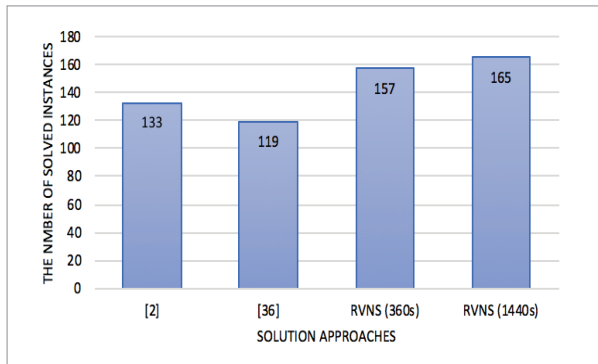
Comparison of different solution approaches on *2CBP*



As shown in Table 8, our RVNS solved the total number of 157 instances belonging to classes 1, 6, 7, 9, and 10 (selected and considered as more challenging in [2] and [36]), outperforming both the number of 113 from [2] and 119 solutions obtained in [36], see Figure 2. Using the time limit of 3600 s as Heßler et al. [19], the authors solved 370 of 400 instances, which is 14 and 5 more instances compared to our RVNS for time limit of 360 s and 1440 s, respectively. However, when it comes to the 50 larger instances from the classes: 1, 4, 5, 9 and 10, our RVNS was more successful in obtaining 31 solutions, compared to 20 solutions from [19], as displayed in Figure 3. On the set of 40 instances with 200 items belonging to classes: 1, 4, 5, and 9, our RVNS solved 22 instances for 360 s and 29 for the time limit of 1440 s, which is less than 35 solutions obtained in [27]. However, having in mind that our RVNS is pri-

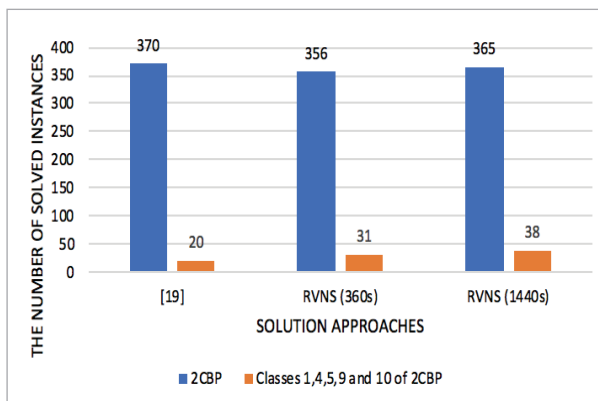
**Figure 2**

Comparison of different solution approaches on instances from classes 1, 6, 7, 9 and 10 of *2CBP*



**Figure 3**

Comparison of our RVNS and [19]



marily designed for solving 2D-Het VBPP, it is overall comparable with the most successful solution methods of 2D-Hom VBPP on *2CBP*.

Another set of instances with larger demands is included in this comparative analysis. Namely, Heßler et al. [19] proposed a set of 400 instances<sup>3</sup> that is obtained from the *2CBP* in the following way. For each item a uniformly random number is generated from the set  $[1, \dots, 100]$  as its frequency, resulting in the instance that has significantly larger number of items. The solution method developed in [19], which was very successful on *2CBP*, showed a poor performance on this new data set, solving only 119 out of 400 instances. Therefore, the authors in [19] developed a special solution method for this type of the problem to solve 324 of 400 instances obtained in such a way. From the available lower and upper bounds for these 400 instances, it can be seen that 339 of them have known optimal solutions, obtained by several solution approaches. Another contribution of our RVNS algorithm is improvement of the obtained results on some of the remaining 61 instances without optimal solution from this new data set. These results are presented in Table 9. Instance's name is contained in the first column, the corresponding number of items in the second, while the next two columns show the known lower and upper bounds of the solution. The objective function value of the new best solution obtained by our RVNS within the time limit of 1 s, 10 s, 60 s, 360 s and 1440 s are represented in the next five columns, respectively. RVNS solutions that improved the corresponding upper bound are bolded.

As it can be seen in Table 9 our RVNS improved the solutions of the 14 instances obtaining smaller objective function value than the corresponding upper bound. Among these instances, the 10 new optimal solutions (marked by "x") are reached, 4 for the time limit of 10 s, 3 more for 60 s and another 3 for 1440 s. For these instances, tests are not repeated with increased time limit after the optimal solution is obtained, i.e. when our RVNS solution coincident with the corresponding lower bound. Therefore, the adaptation of our RVNS algorithm for 2D-HomVBPP made another contribution.<sup>4</sup>

<sup>3</sup> <https://logistik.bwl.uni-mainz.de/forschung/benchmarks/>

<sup>4</sup> for new solutions on these 14 instances see <https://doi.org/10.5281/zenodo.5321272>



**Table 9**

Comparison of results on large instances

<i>Instance</i>	<i>n(items)</i>	<i>LB</i>	<i>UB</i>	<b>RVNS 1s</b>	<b>RVNS 10s</b>	<b>RVNS 60s</b>	<b>RVNS 360s</b>	<b>RVNS 1440s</b>
CL_04_100_06	5089	627	628	691	628	<b>627</b> *	/	/
CL_04_100_08	5089	642	645	687	<b>643</b>	<b>642</b> *	/	/
CL_04_200_01	10229	1293	1297	1453	1373	1308	<b>1294</b>	<b>1293</b> *
CL_05_100_06	5089	314	315	322	<b>314</b> *	/	/	/
CL_05_100_08	5089	321	323	<b>322</b>	<b>321</b> *	/	/	/
CL_05_100_10	5089	327	328	333	<b>327</b> *	/	/	/
CL_05_200_02	10141	624	629	682	633	<b>628</b>	<b>627</b>	<b>627</b>
CL_05_200_03	10157	630	635	708	646	<b>634</b>	<b>633</b>	<b>633</b>
CL_05_200_04	10141	630	631	670	635	631	631	<b>630</b> *
CL_05_200_05	10141	632	640	704	646	<b>634</b>	<b>633</b>	<b>632</b> *
CL_05_200_06	10141	626	630	686	630	<b>627</b>	<b>627</b>	<b>627</b>
CL_05_200_07	10157	630	636	715	644	636	<b>635</b>	<b>634</b>
CL_05_200_08	10141	635	640	681	<b>635</b> *	/	/	/
CL_05_200_10	10141	632	635	690	<b>633</b>	<b>632</b> *	/	/

## 4. Conclusion

The metaheuristic RVNS (Variable Neighborhood Search) based algorithm was used to solve the 2D vector bin packing problem, in its heterogeneous 2D-HetVBPP and homogeneous 2D-HomVBPP variant, because the exact solution using standard mathematical model and CPLEX solver is not possible for larger instances, and the approximate solutions obtained in limited time are of a poor quality. The set of 6 small 2D-HetVBPP instances (with up to 20 items) and the 50 large 2D-HetVBPP instances (with up to 1000 items) was prepared and used to test the efficiency of the algorithm. The specialized version of the algorithm solved some new instances (compared to results in [2], [6], [10], [19], [27], [36]) of the bench-

mark set 2CBP [8] and the set of instances with larger demands [19]. More precisely,

- on 400 instances of the data set 2CBP [8], our solution approach solved 356 for 360s and 365 for 1440s of running time,
- 400 instances of the data set from [19], with 61 unsolved instances with the known upper and lower bounds, we provided 10 new optimal solutions and 4 new upper bounds.

## Acknowledgement

The authors are thankful to anonymous referees for insightful comments that contributed to the improvement of the paper.

## References

1. Alves, C., de Carvalho, J. V., Clautiaux, F., Rietz, J. Multidimensional Dual-Feasible Functions and Fast Lower Bounds for the Vector Packing Problem. *European Journal of Operational Research*, 2014, 233(1), 43-63. <https://doi.org/10.1016/j.ejor.2013.08.011>
2. Aringhieri, R., Duma, D., Grosso, A., Hosteins, P. Simple but Effective Heuristics for the 2-Constraint Bin Packing Problem. *Journal of Heuristics*, 2018, 24(3), 345-357. <https://doi.org/10.1007/s10732-017-9326-0>

3. Baker, B. M., Ayechev, M. A. A Genetic Algorithm for the Vehicle Routing Problem. *Computers & Operations Research*, 2003, 30, 787-800. [https://doi.org/10.1016/S0305-0548\(02\)00051-5](https://doi.org/10.1016/S0305-0548(02)00051-5)
4. Billaut, J. C., Della Croce, F., Grosso, A. A Single Machine Scheduling Problem with Two-Dimensional Vector Packing Constraints. *European Journal of Operational Research*, 2015, 243(1), 75-81. <https://doi.org/10.1016/j.ejor.2014.11.036>
5. Bortfeldt, A., Wäscher, G. Constraints in Container Loading-A State-of-the-Art Review. *European Journal of Operational Research*, 2013, 229(1), 1-20. <https://doi.org/10.1016/j.ejor.2012.12.006>
6. Brandao, F., Pedroso, J. P. Bin Packing and Related Problems: General Arc-Flow Formulation with Graph Compression. *Computers & Operations Research*, 2016, 69, 56-67. <https://doi.org/10.1016/j.cor.2015.11.009>
7. Buljubašić, M., Vasquez, M. Consistent Neighborhood Search for One-Dimensional Bin Packing and Two-Dimensional Vector Packing. *Computers & Operations Research*, 2016, 76, 12-21. <https://doi.org/10.1016/j.cor.2016.06.009>
8. Caprara, A., Toth, P. Lower Bounds and Algorithms for the 2-Dimensional Vector Packing Problem. *Discrete Applied Mathematics*, 2001, 111(3), 231-262. [https://doi.org/10.1016/S0166-218X\(00\)00267-5](https://doi.org/10.1016/S0166-218X(00)00267-5)
9. Christensen, H. I., Khan, A., Pokutta, S., Tetali, P. Approximation and Online Algorithms for Multidimensional Bin Packing: A Survey. *Computer Science Review*, 2017, 24, 63-79. <https://doi.org/10.1016/j.cosrev.2016.12.001>
10. Gabay, M., Zaourar, S. Vector Bin Packing with Heterogeneous Bins: Application to the Machine Reassignment Problem. *Annals of Operations Research*, 2016, 2-16, 242(1), 161-194. <https://doi.org/10.1007/s10479-015-1973-7>
11. Garey, M. R., Graham, R. L., Johnson, D. S., Yao, A. C. C. Resource Constrained Scheduling as Generalized Bin Packing. *Journal of Combinatorial Theory Series A*, 1976, 21(3), 257-298. [https://doi.org/10.1016/0097-3165\(76\)90001-7](https://doi.org/10.1016/0097-3165(76)90001-7)
12. Garey, M. R., Johnson, D. S. *Computers and Intractability: A Guide to Np-Completeness*, 1979.
13. Han, B. T., Diehr, G., Cook, J. S. Multiple-Type, Two-Dimensional Bin Packing Problems: Applications and Algorithms. *Annals of Operations Research*, 1994, 50(1), 239-261. <https://doi.org/10.1007/BF02085642>
14. Hansen, P., Mladenović, N. Variable Neighborhood Search: Principles and Applications. *European Journal of Operational Research*, 2001, 130(3), 449-467. [https://doi.org/10.1016/S0377-2217\(00\)00100-4](https://doi.org/10.1016/S0377-2217(00)00100-4)
15. Hansen, P., Mladenović, N. Variable Neighborhood Search. In: Burke, E. K., Graham, R. D. (Eds.) *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Springer-Verlag, New York, 2014, 313-337. [https://doi.org/10.1007/978-1-4614-6940-7\\_12](https://doi.org/10.1007/978-1-4614-6940-7_12)
16. Hansen, P., Mladenović, N., Pérez, J. A. Variable Neighbourhood Search: Methods and Applications. *Annals of Operations Research*, 2010, 175(1), 367-407. <https://doi.org/10.1007/s10479-009-0657-6>
17. Hansen, P., Mladenović, N., Todosijević R., Hanafi, S. Variable Neighborhood Search: Basics and Variants. *EURO Journal on Computational Optimization*, 2017, 5(3), 423-454. <https://doi.org/10.1007/s13675-016-0075-x>
18. Hansen, P., Mladenović, N., Brimberg, J., Pérez, J. A. M. Variable Neighborhood Search. In *Handbook of metaheuristics*, 2019, Springer, Cham, 57-97. [https://doi.org/10.1007/0-306-48056-5\\_6](https://doi.org/10.1007/0-306-48056-5_6)
19. Heßler, K., Gschwind, T., Irnich, S. Stabilized Branch-and-Price Algorithms for Vector Packing Problems. *European Journal of Operational Research*, 2018, 271(2), 401-419. <https://doi.org/10.1016/j.ejor.2018.04.047>
20. Heßler, K., Irnich, S., Kreiter, T., Pferschy, U. Bin Packing with Lexicographic Objectives for Loading Weight-and Volume-Constrained Trucks in a Direct-Shipping System. *OR Spectrum*, 2021, 1-43. <https://doi.org/10.1007/s00291-021-00628-x>
21. Hu, Q., Wei, L., Lim, A. The Two-Dimensional Vector Packing Problem with General Costs. *Omega*, 2018, 74, 59-69. <https://doi.org/10.1016/j.omega.2017.01.006>
22. Masson, R., Vidal, T., Michallet, J., Penna, P. H. V., Petrucci, V., Subramanian, A., Dubedout, H. An Iterated Local Search Heuristic for Multi-Capacity Bin Packing and Machine Reassignment Problems. *Expert Systems with Applications*, 2013, 40(13), 5266-5275. <https://doi.org/10.1016/j.eswa.2013.03.037>
23. Mladenović, N., Hansen, P. Variable Neighborhood Search. *Computers & Operations Research*, 1997, 24(11), 1097-1100. [https://doi.org/10.1016/S0305-0548\(97\)00031-2](https://doi.org/10.1016/S0305-0548(97)00031-2)
24. Munien, C., Ezugwu, A. E. Metaheuristic Algorithms for One-Dimensional Bin-Packing Problems: A Survey of Recent Advances and Applications. *Journal of Intelligent Systems*, 2021, 30(1), 636-663. <https://doi.org/10.1515/jisys-2020-0117>

25. Notteboom, T. E. Container Shipping and Ports: An Overview. *Review of network economics*, 2004, 3(2), 86-106. <https://doi.org/10.2202/1446-9022.1045>
26. Panigrahy, R., Talwar, K., Uyeda, L., Wieder, U. Heuristics for Vector Bin Packing. Research. Microsoft. com. Technical report, Microsoft Research, 2011
27. Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F. A Generic Exact Solver for Vehicle Routing and Related Problems. *Mathematical Programming*, 2020, 183(1), 483-523. <https://doi.org/10.1007/s10107-020-01523-z>
28. Pessoa, A., Sadykov, R., Uchoa, E. Solving Bin Packing Problems Using VRPSolver Models. In *Operations Research Forum*, 2021, (Vol. 2, No. 2, pp. 1-25). Springer International Publishing. <https://doi.org/10.1007/s43069-020-00047-8>
29. Rajković, R., Zrnić, N., Stakić, Đ., Sedmak, A., Kirin, S. An Approach to Determine Optimal Number of Containers for Cargo Stacking in Function of Transportation Cost. In: *Proceedings - 6th International Symposium on Industrial Engineering - SIE 2015, 24th-25th September 2015, Belgrade, Serbia*, 300-303.
30. Ramos-Figueroa, O., Quiroz-Castellanos, M., Mezura-Montes, E., Schütze, O. Metaheuristics to Solve Grouping Problems: A Review and a Case Study. *Swarm and Evolutionary Computation*, 2020, 53, 100643. <https://doi.org/10.1016/j.swevo.2019.100643>
31. Santos, L. F. M., Iwayama, R. S., Cavalcanti, L. B., Turi, L. M., de Souza Morais, F. E., Mormilho, G., Cunha, C. B. A Variable Neighborhood Search Algorithm for the Bin Packing Problem with Compatible Categories. *Expert Systems with Applications*, 2019, 124, 209-225. <https://doi.org/10.1016/j.eswa.2019.01.052>
32. Shachnai, H., Tamir, T. Approximation Schemes for Generalized Two-Dimensional Vector Packing with Application to Data Placement. *Journal of Discrete Algorithms*, 2012, 10, 35-48. <https://doi.org/10.1016/j.jda.2011.07.001>
33. Spieksma F. C. R. A Branch-and-Bound Algorithm for the Two-Dimensional Vector Packing Problem. *Computers & Operations Research*, 1994, 21(1), 19-25. [https://doi.org/10.1016/0305-0548\(94\)90059-0](https://doi.org/10.1016/0305-0548(94)90059-0)
34. Stakić, Đ., Anokić, A., Jovanović, R. Comparison of Different GRASP Algorithms for the Heterogeneous Vector Bin Packing Problem. In: *2019 China-Qatar International Workshop on Artificial Intelligence and Applications to Intelligent Manufacturing (AIAIM)*, IEEE, 2019, 63-70. <https://doi.org/10.1109/AIAIM.2019.8632779>
35. Stakić, Đ., Živković, M., Anokić, A., Rajković, R. Solving the Problem of Packing Packages in the Containers with the Limitation of Mass and the Volume by VNS Method. In: *XLV Symposium on Operational Research, Zlatibor, Serbia, September 2018*, 112-117.
36. Turkey, A., Sabar, N. R., Dunstall, S., Song, A. Hyper-Heuristic Local Search for Combinatorial Optimisation Problems. *Knowledge-Based Systems*, 2020, 205, 106-264. <https://doi.org/10.1016/j.knosys.2020.106264>
37. Wei, L., Lai, M., Lim, A., Hu, Q. A Branch-and-Price Algorithm for the Two-Dimensional Vector Packing Problem. *European Journal of Operational Research*, 2020, 281(1), 25-35. <https://doi.org/10.1016/j.ejor.2019.08.024>

