Heuristics for Variable Cost and Size Cluster Vector Bin Packing

Laura Wolf[1](✉), Sabrina Klos[2], and Stefan Nickel[1]

[1] Karlsruhe Institute of Technology, Institute for Operations Research, 76131 Karlsruhe, Germany, {laura.wolf, stefan.nickel}@kit.edu
[2] Technical University of Applied Sciences Würzburg-Schweinfurt, 97070 Würzburg, Germany, sabrina.klos@thws.de

Abstract. The Variable Cost and Size Cluster Vector Bin Packing (VCSCVBP) problem involves assigning items of different sizes to bins of various sizes that are grouped into clusters, with each cluster incurring specific costs. These clusters may contain homogeneous or heterogeneous bins, reflecting real-world applications in domains such as cloud computing and logistics. Prior research has focused on heuristics for clusters of homogeneous bins. In this work, we extend the investigation to problem settings with clusters of heterogeneous bins. Specifically, we propose heuristics tailored to VCSCVBP based on a variable neighborhood search and a genetic algorithm. We benchmark the performance of a variable neighborhood search, a genetic algorithm, and a previously published constructive heuristic, called the Combined CS-P heuristic, against an exact solver-based solution method. For this purpose, we generate novel benchmark data sets for the VCSCVBP problem. Extensive computational experiments demonstrate that, on average, the constructive Combined CS-P heuristic achieves a high solution quality. However, the genetic algorithm offers significant performance improvements over existing techniques for certain types of clusters with heterogeneous bins and cost structures. These findings underscore the potential of metaheuristic approaches for effectively addressing VCSCVBP instances.

Keywords: Vector Bin Packing · Variable Neighborhood Search · Genetic Algorithm

1 Introduction

In this paper, we investigate heuristics for the Variable Cost and Size Cluster Vector Bin Packing (VCSCVBP) problem [15]. The constructive CS-P heuristics previously proposed for VCSCVBP in [15] demonstrate strong performance for clusters of homogeneous bins, wherein all bins within a cluster share identical characteristics (i.e., uniform dimensions). However, as we demonstrate in this study, the performance of the CS-P heuristics is constrained for specific configurations involving clusters with heterogeneous bins. To address this limitation, we propose two metaheuristic approaches: a Variable Neighborhood Search (VNS) and a Genetic Algorithm (GA). In this section, we first provide an introduction

to Vector Bin Packing (VBP) and formally present the VCSCVBP problem. Subsequently, we review the CS-P heuristics and the corresponding metrics. Finally, we delineate the main contributions of this paper.

VBP extends the Bin Packing (BP) problem, which involves packing items into bins of a single dimension. The d-dimensional VBP problem considers a finite item set J, where each item j possesses size w_{jk} along multiple dimensions $k \in K$ ($1 \leq k \leq d$), represented as a vector. These items must be packed into a bin i from bin set I while respecting capacity constraints W_{ik} of the bin. The literature sometimes refers to VBP as the multi-dimensional bin packing problem [10]. In [15], we extended VBP to VCSCVBP to incorporate a cluster structure of bins with various bin sizes and cluster costs. This work builds upon existing research in the fields of BP and VBP, including the Variable Cost and Size Bin Packing Problem (VCSBPP) [4], Vector Bin Packing with Heterogeneous Bins (VBPHB) [5], and the Temporal Bin Packing Problem (TBPP) with Placement Constraints [1]. The latter also considers a cluster structure, but without costs, where the clusters are called racks due to the application in cloud computing.

In the context of the VCSCVBP problem [15], let I denote the finite set of bins, where each bin $i \in I$ is characterized by its capacity W_{ik} across multiple dimensions $k \in K$, as well as by upper bounds f_{ik} on the fill levels in those dimensions. Furthermore, let G represent the finite set of clusters, where each cluster $g \in G$ comprises a specified subset of bins I_g^G and is associated with a predetermined cost parameter c_g. It is assumed that each bin is assigned to exactly one cluster. Additionally, let J denote the finite set of items to be packed, each item $j \in J$ being described by its size w_{jk} in dimension $k \in K$. The objective is to minimize the total costs incurred by cluster utilization while assigning all items to bins. This requires making binary decisions about the activation of specific clusters (x_g) and bins (y_i), as well as determining the allocation of items to bins (z_{ij}), subject to capacity constraints. The objective function (1) minimizes total costs by summing the costs across all activated clusters. Constraints (2) and (3) ensure that activating any bin within a cluster necessitates the activation of the entire cluster. Constraints (4) guarantee that each item is assigned to exactly one bin. Constraints (5) enforce the capacity limits of the bins for each dimension $k \in K$, while respecting the maximum fill level defined by f_{ik}.

VCSCVBP:

$$\min \quad Z = \sum_{g \in G} c_g x_g \tag{1}$$

$$\text{s.t.} \quad x_g \leq \sum_{i \in I_g^G} y_i \qquad g \in G \tag{2}$$

$$x_g \geq y_i \qquad g \in G, i \in I_g^G \tag{3}$$

$$\sum_{i \in I} z_{ij} = 1 \qquad j \in J \tag{4}$$

$$\sum_{j \in J} w_{jk} z_{ij} \leq f_{ik} W_{ik} y_i \qquad i \in I, k \in K \tag{5}$$

$$x_g, y_i, z_{ij} \in \{0,1\} \qquad g \in G, i \in I, j \in J \tag{6}$$

VCSCVBP can be applied to various planning problems. In the field of cloud capacity planning, clusters can be seen as a grouping of hosts, and the items are the virtual machines (VMs) to be placed on the hosts [15]. Given the cloud computing use case, we include the maximum fill level f_{ik} in the VCSCVBP model to capture technical constraints. Another application concerns packing with predefined compartments, where vehicles with different compartment configurations are available, and the selection of the vehicles needs to be optimized.

Next, we present publications with relevant work in the field of heuristics for BP and VBP. For the VSBPP, [6] introduces constructive heuristics, a set covering-based heuristic, and a GA, while [7] presents a VNS. [12] and [13] provide comprehensive overviews of heuristics for VBP. Some of these algorithms have been further developed into multiple constructive CS-P heuristics for VCSCVBP in [15]. The CS-P heuristics comprise the following components: a cluster selection step, which determines which clusters to fill next; a packing step, which assigns items to bins; and a post-processing step, which may change the last cluster filled. Various VBP algorithms can be employed to select the next item to be placed, including First Fit Decreasing (FFD) [10], Norm-based Greedy (NBG) [14], Dot Product (DP) [14], Local Search (LS) [13], and Hybrid Heuristics (HyL_2, HyDP) [13]. In [15], we incorporate CS-P heuristics with different packing algorithms in the Combined CS-P heuristic to identify the best solution across all CS-P heuristic variants. In the cluster selection step, the decision which cluster to fill next is determined by selecting the cluster with the lowest cost-to-combined-size ratio (c/v-ratio) among those that can accommodate the considered item(s).

At this point, we provide an overview of the concepts of combined size v and the c/v-ratio, as they are employed in heuristics for VBP and in the CS-P heuristics for VCSCVBP for item, bin, and cluster selection. These concepts also play a crucial role in the VNS and GA for VCSCVBP presented subsequently, as well as in the determination of cluster costs in the experimental setup. The combined size v aggregates an item's or bin's sizes across multiple resource dimensions into one single metric for sorting. It can be calculated using various norms, and the different resource dimensions may be weighted with a factor g_k for each dimension $k \in K$. Eq. (7) and Eq. (8) specify the calculation of the combined size v_i for bin $i \in I$ based on the bin size W_{ik} and of the combined size v_j for an item $j \in J$ based on the item size w_{jk}, applying the L_2-norm:

$$v_i = \sqrt{\sum_{k \in K} g_k W_{ik}^2} \tag{7}$$

$$v_j = \sqrt{\sum_{k \in K} g_k w_{jk}^2} \tag{8}$$

Distinct weighting schemes can be employed to account for differences among dimensions, such as unit weights or based on the item sizes w_{jk} reciprocal average weights. For an overview on norms and weight definitions, refer to [12]. In [15], for VCSCVBP, we aggregate the combined sizes v_i for each bin in a cluster and obtain the combined size v_g of the cluster. To select the next cluster to fill in the CS-P heuristics, we define the c/v-ratio as a metric based on the combined size v_g of the cluster and its costs c_g.

Numerical evaluation in [15] demonstrated that the Combined CS-P heuristic offers high solution quality with lower average runtimes than the solver-based solution method on a data set based on cloud computing data with clusters of homogeneous bins. In this paper, we detail the limitations of the Combined CS-P heuristic for clusters of heterogeneous bins and propose algorithms tailored to more diverse cluster structures. The main contributions of this paper can be summarized as follows:

- We propose a VNS and a GA specifically tailored to the VCSCVBP problem.
- We generate a novel data set for VCSCVBP by combining established VBP benchmarks with various cluster configurations, including bin types and cost structures.
- We conduct extensive computational experiments to compare the performance of the presented heuristics across different cluster configurations and cost structures.

Accordingly, Sect. 2 outlines the VNS and the GA for VCSCVBP. In Sect. 3, we first describe the data set generation. Next, we analyze the performance of the heuristics for clusters of homogeneous and heterogeneous bins. Section 4 summarizes the contributions and outlines further research directions.

2 Variable Neighborhood Search and Genetic Algorithm for VCSCVBP

In this section, we present the VNS and GA, which we adapt from the BP and VBP literature to address the VCSCVBP problem. To avoid repetition in Sect. 3, we specify the parametrization used for the computational experiments here, though alternative parameterizations are possible.

2.1 Variable Neighborhood Search for VCSCVBP

The metaheuristic VNS is initially delineated in [11]. A local search is combined with a procedure for changing neighborhoods to prevent entrapment in a local optimum. Our approach is based on the VNS for the variable-sized bin packing problem introduced by [7]. We adapt the procedure, specifically the shaking and the local search procedures, to a VBP problem and a cluster structure of bins.

The detailed procedure of Algorithm 1 of the VNS for VCSCVBP is explained step by step in the following. A solution consists of all used clusters and bins,

as well as information on the bin assignment of each item. We begin by creating an initial solution using the constructive Combined CS-P heuristic presented in [15]. Initially, the weight of each neighborhood selection strategy c_k is set to 1 (see line 2 of Algorithm 1). A neighborhood structure $\mathcal{N}_k$ to be explored next is chosen with a likelihood proportional to the weights c_k (see line 4 of Algorithm 1). The set of solutions in the k^{th} neighborhood of x is defined as $\mathcal{N}_k(x)$. Then, a new solution $x' \in \mathcal{N}_k(x)$ is created using the k^{th} neighborhood of x (see line 5 of Algorithm 1). Subsequently, a local search is performed (see line 6 of Algorithm 1). The neighborhood selection strategies and the local search methods are described in detail below after the algorithm overview. If the resulting solution x'' attains a lower objective value than the best solution found so far, the best solution found is updated, and the weight of the chosen neighborhood selection strategy is increased by 1 (see lines 7 to 9 of Algorithm 1). Thus, successful neighborhood selection strategies are more frequently chosen in future iterations. The algorithm is executed until a termination condition is met. This can be a maximum number of runs or a time limit. In our implementation, the overall number of iterations is set to 10.

Algorithm 1 Variable neighborhood search

```
1: bestSolution ← ConstructiveHeuristic()
2: c_1 = 1, c_2 = 1
3: while not TerminationConditionMet() do
4:     k ← ChooseNeighborhoodStructure(c_1, c_2)
5:     x' ← Shaking(bestSolution, k)
6:     x'' ← LocalSearch(x')
7:     if x'' has lower objective value than bestSolution then
8:         bestSolution ← x''
9:         c_k ← c_k + 1
10:    end if
11: end while
12: return bestSolution
```

The neighborhood selection methods (see line 5 of Algorithm 1) used for shaking to generate a neighborhood structure are explained in the following:

- Random reassignment of cluster types for a subset of clusters ($\mathcal{N}_1(x)$): In this procedure, a fixed proportion of clusters is assigned a new cluster type chosen at random. In our study, we set this fixed proportion at 0.2. All items within each reassigned cluster are then sorted in descending order according to their combined size v_j. For each item, placement is attempted in the cluster with the reassigned type. If such placement is not feasible due to capacity constraints, the algorithm attempts to accommodate the item in another existing cluster. If neither of these options is possible, a new cluster is selected that can accommodate the item. The probability of opening a new

cluster is inversely proportional to its c/v-ratio, which means that clusters with smaller c/v-ratios have a higher likelihood of being opened.
- Deletion of a specified share of clusters ($\mathcal{N}_2(x)$): A predefined fraction of clusters is chosen not to be used. In our experiments, we set this value to 0.2. The items previously placed into bins of one of these clusters are reallocated according to the procedure as described above. They are first sorted by their combined size v_j in decreasing order. Each item is then considered for placement into bins of existing clusters. If the item cannot be accommodated in any bin of the remaining clusters, a new cluster is opened with a probability that is inversely proportional to its c/v-ratio.

The implemented local search procedures (see line 6 of Algorithm 1) are listed below. The aim is to increase cluster density so that, in the end, clusters can be removed by reassigning items under the condition that the new placement is feasible. The procedures are applied in the following order:

- Reassignment of items to improve capacity utilization: Items are moved from a bin of the actual cluster to a bin of another used cluster if, as a result, the remaining capacity in terms of the remaining combined size v_g of the receiving cluster is less than the remaining combined size v_g of the original cluster prior to the move.
- Swapping of items: Items are exchanged between bins under the same conditions as in the reassignment procedure. A swap is performed if, following the exchange, the remaining combined size v_g of one cluster is less than the remaining combined size v_g of the other cluster prior to the swap. In this case, items are swapped between the bins belonging to the two clusters.
- Reassignment of items from sparsely filled clusters: First, the cluster loads are calculated for each cluster with the help of the combined size v_g. Then, all items placed into bins of the least filled cluster are tried to be placed into bins of other clusters. If that is possible, the empty cluster is removed.

The VNS for VCSCVBP is evaluated in Sect. 3, where it is compared to the other heuristics outlined.

2.2 Genetic Algorithm for VCSCVBP

A GA is a metaheuristic based on natural selection [9]. A population undergoes the steps of selection, crossover, and mutation, aiming to ensure that the best possible candidate solutions of the generation are passed on to the next generation. Algorithm 2 illustrates the general GA procedure. Our approach is based on [2], which introduces a GA for the variable-sized VBP problem with a limited number of bins. A major distinction in our problem setting lies in the incorporation of the cluster structure of bins, which is why we adapt the algorithm. Our approach applies to both a limited and an unlimited number of clusters.

First, we describe the chromosome representation, including the fitness function, before explaining the crossover and mutation operators. Specifying exactly

Algorithm 2 Genetic algorithm

```
1: population ← InitializePopulation(populationSize)
2: generation ← 0
3: EvaluateFitness(population)
4: while generation < maxGenerations and not TerminationConditionMet
   do
5:     newPopulation ← ∅
6:     while |newPopulation| < populationSize do
7:         parent1 ← SelectParent(population)
8:         parent2 ← SelectParent(population)
9:         child ← Crossover(parent1, parent2)
10:        child ← Mutate(child)
11:        newPopulation ← newPopulation ∪ {child}
12:    end while
13:    population ← newPopulation
14:    EvaluateFitness(population)
15:    generation ← generation + 1
16: end while
17: return BestIndividual(population)
```

which item is packed into which bin of a cluster would introduce significant redundancy among the solutions, as several items of the same item type and clusters of the same cluster type share the same specifications. Consequently, we assign each item only a cluster type, with the following encoding: The items are sorted in a vector, (j_1,..., $j_{|J|}$), by their combined size v_j in descending order. The second vector, (h_1,..., $h_{|J|}$), represents the chromosome. Each entry in the second vector corresponds to a cluster type assignment of the item at this position in the first vector. For example, item j_1 is placed into a bin of cluster type h_1. [2] demonstrates that their Plain-FFD encoding already produces favorable results compared to a more detailed encoding. Therefore, we refrain from introducing any further encoding. In contrast to our encoding, they use bin types instead of cluster types.

For evaluation, the following procedure is applied (see lines 3 and 14 of Algorithm 2): An FFD algorithm is used independently for all items assigned to a specific cluster type. First, one cluster of this type is opened, and the items assigned to the cluster type are packed using the FFD algorithm for VBP described in [10]. In the case of clusters of heterogeneous bins, the bins of one cluster are sorted in ascending order by their combined size v_i. If an item cannot be placed in any of the bins of the first cluster, a new cluster is opened, and so forth. This procedure is carried out independently for each cluster type to determine the required number of clusters of each type.

The fitness function (9) is based on [2] and consists of three components. The first part represents the total cluster costs, defined as the sum of the individual cluster costs of all utilized clusters. In the case of a limited number of clusters,

the second part consists of the sum of the penalties c_j^l for each item $j \in J$, which cannot be placed in the determined cluster type. This penalty term for each item should be chosen higher than the sum of all possible cluster costs, as it reflects an infeasible solution. For an unlimited number of clusters per cluster type, this term can be neglected. The third component of the fitness function concerns the density of the packing. Densely packed clusters are preferred. Specifically, we calculate the overall used resources v_i^{used} for each bin $i \in I$ and the available capacity v_i, using the L_2-norm and the weights as introduced in Eq. 7. The overall bin resources W_{ik} contribute to computing v_i, while only the used bin resources are considered when determining v_i^{used}. The packing density is then the ratio of v_i^{used} to v_i. The overall packing density is multiplied by the factor φ. The influence of density in the fitness function is designed only to provide additional support so as not to exceed the influence of the cluster costs. For this reason, we set φ to be 0.9 times the cost of the cluster type with the minimal cost. The fitness function, which integrates the three components described above, is defined as

$$F = \sum_{g \in G} c_g x_g + \sum_{j \in J} (1 - \sum_{i \in I} z_{ij}) c_j^l - \varphi \sum_{i \in I} \frac{v_i^{used}}{v_i}. \tag{9}$$

Initially, all possible cluster types for each item are determined. This is done to ensure that items fit into any unfilled bin of the selected cluster. Next, an initial population is created randomly. For each item, one feasible cluster type is selected with equal probability. For the selection procedure, we apply tournament selection. Thereby, for each parent, we randomly choose 30% of the population and select the chromosome with the best fitness out of this population.

For crossover, we use one-point, two-point, and three-point crossovers (see line 9 of Algorithm 2). For a one-point crossover, one crossover point is selected at random. The cluster types preceding the crossover point are copied from the first parent to the same position of the child chromosome. The positions after the crossover point are filled with the cluster types of the second parent at the same position. The one-point crossover is applied with a probability of 0.4. For two-point crossover and three-point crossover, each with a probability of 0.3, the procedure is analogous. Two or three crossover points are determined randomly, and the segments are then filled alternately from the first and second parents at the same position. In cases where crossover assigns a cluster type to an item that can not accommodate the item, another feasible cluster type is randomly selected for the item. The same logic applies to the mutation operators, described as follows (see line 10 of Algorithm 2):

- Random reassignment of cluster types: For each item, the cluster type is changed with a probability of 0.7. This mutation operator is applied with a probability of 0.6.
- Random relocation of items from a selected cluster type: A cluster type is selected randomly. Each item assigned to this cluster type is moved to another randomly chosen cluster type with a probability of 0.5. This mutation operator is applied with a probability of 0.2.

– Complete reassignment of items of a cluster type: All items associated with a randomly selected cluster type are reassigned to another cluster type. This mutation operator is chosen with a probability of 0.2.

In the computational experiments, we set the population size to 100 and run the algorithm for 50 generations.

3 Computational Experiments

In this section, we conduct computational experiments to compare the heuristics for VCSCVBP for clusters of both homogeneous and heterogeneous bins. The constructive Combined CS-P heuristic, the VNS, and the GA are compared to the exact solver-based solution, and their benefits in specific problem settings are demonstrated. The models and algorithms are implemented in Python 3.10. Gurobi 11.3.0 is used as a solver. A MIP gap of 0.01% is applied. All computations are conducted with a time limit of 3600 seconds. The computational experiments are executed on a compute server with 1 TB of RAM equipped with Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40 GHz processors with 14 cores.

3.1 Cluster Vector Bin Packing Data Sets

We extend an existing VBP data set from the literature to the VCSCVBP data sets, as the mathematical models proposed in this work are based on VBP formulations. [3] introduces 10 different instance classes, each with a demand of one per item size. [8] extends these instances by utilizing the instance classes and item sizes by [3], but drawing the demand for each item size from a discrete uniform distribution with values between 1 and 100. We further adapt this extended data set as follows to achieve different bin sizes and item sizes with a larger difference in their dimensions in the same instance:

We combine three instance classes from [8] into one new instance class for the item demand, as shown in Table 1. For example, our instance class $B1$ comprises the instance classes *CL_01_25*, *CL_02_25*, and *CL_05_25*. As a result of the combination of three instance classes with 25 item types each, we obtain 75 different item types. [8] generates 10 instances per instance class. We aggregate the first instance from each class, the second instance from each class, and so forth. By creating four new instance classes with 10 instances each, we consider 40 demand scenarios overall. To reduce the number of items per item type, as we increase the number through combination, we use the demand of each item class divided by 10 and rounded. This results in 375 items per instance.

We introduce two VCSCVBP data sets: The VCSCVBP data set with clusters of homogeneous bins (VCSCVBP_HO) and the one with clusters of heterogeneous bins (VCSCVBP_HE). Table 2 presents the defined cluster types, bin capacities, and cost structures for a cluster structure with homogeneous bins. Table 3 displays the cluster types, bin capacities, and cost structures for cluster types with heterogeneous bins. For example, cluster *Cl_B_4* comprises three

Table 1. Classes of problem instances in the VCSCVBP data sets

Class	Number of items	Class 1		Class 2		Class 3	
		Class name	Item dim.	Class name	Item dim.	Class name	Item dim.
B1	375	CL_01_25	[100, 400]	CL_02_25	[1, 1000]	CL_05_25	[25, 100]
B2	375	CL_01_25	[100, 400]	CL_02_25	[1, 1000]	CL_06_25	[20, 100]
B3	375	CL_01_25	[100, 400]	CL_03_25	[200, 800]	CL_05_25	[25, 100]
B4	375	CL_01_25	[100, 400]	CL_03_25	[200, 800]	CL_06_25	[20, 100]

bins of bin type 1 with the dimensions 200 x 200, three bins of type 2 with the dimensions 500 x 500, and four bins of type 3 with the dimensions 1000 x 1000. The number of bins per cluster is set to 10. For each instance, 40 clusters of each cluster type are defined as the upper bound to be chosen. To select the cluster costs, we use the c/v-ratio as a metric and thus reflect different cost developments with increasing bin capacity.

Table 2. Cluster and bin configurations in the VCSCVBP_HO data set

Cluster type	Cl_B_1	Cl_B_2	Cl_B_3
Bin capacity in k_1	200	500	1000
Bin capacity in k_2	200	500	1000
Bins per Cluster	10	10	10
c/v-ratio in cost structure (a)	0.8	0.9	1
c/v-ratio in cost structure (b)	1.2	1.1	1
c/v-ratio in cost structure (c)	0.4	0.6	1

Table 3. Cluster and bin configurations in the VCSCVBP_HE data set

Cluster type	Cl_B_4	Cl_B_5	Cl_B_6
Bin capacity in k_1	[200, 500, 1000]	[200, 500, 1000]	[200, 500, 1000]
Bin capacity in k_2	[200, 500, 1000]	[200, 500, 1000]	[200, 500, 1000]
Bins per Cluster	[3, 3, 4]	[4, 6, 0]	[8, 0, 2]
c/v-ratio in cost structure (d)	0.8	0.9	1
c/v-ratio in cost structure (e)	1.2	1.1	1
c/v-ratio in cost structure (f)	0.4	0.6	1

Since the items in each dimension are within the same range, we use uniform weights. To ensure comparability with the existing data sets, we adopt a filling level of 1 ($f_{k_1} = 1$ and $f_{k_2} = 1$) in both dimensions. With 40 demand scenarios and three cost structures each, we consider 120 instances for clusters of homogeneous bins and 120 instances for clusters of heterogeneous bins.

3.2 Heuristics for Clusters of Homogeneous Bins

This section provides an overview of the computational results for clusters of homogeneous bins and compares the Combined CS-P heuristic, VNS, and GA with the solver-based solution method. For all subsequent computational experiments described, we use the Combined CS-P heuristic from [15], combining all CS-P heuristics into one by calculating the objective value of each of the six CS-P heuristics independently and selecting the best objective value. We additionally include an extension for experiments with the VCSCVBP_HO data set. We systematically exclude cluster types from the heuristic in sequence, starting with the cluster containing the smallest bins, to evaluate the trade-off between the additional packing options of larger bins and the increased c/v-ratio. The exclusion proceeds in three stages: first, all cluster types are considered; next, only cluster types *Cl_B_2* and *Cl_B_3* are considered; and finally, only *Cl_B_3* is considered. For the post-processing step, all cluster types are reconsidered. For the VNS, the solution of the Combined CS-P heuristic without extension is used as the initial solution. We compare the performance of the Combined CS-P heuristics without and with extension, VNS, and GA, with the solver-based solution method for the VCSCVBP_HO data set. The performance metric ΔZ describes the average deviation of the objective value of the best solution found by the heuristic from that of the solver in relation to the latter. The metric Δt outlines the mean runtime difference as the runtime of the heuristic subtracted from that of the solver.

Table 4 displays results for the VCSCVBP_HO data set. Due to the large problem instances, the heuristics often perform better within the time limit than the solver, resulting in negative values for ΔZ. Thus, we do not include the number of instances for which the heuristics found the optimal solution, because the solver found the optimal solution for no instance or only a small number of instances in both data sets. Comparing the Combined CS-P heuristic without and with extension, the effect of excluding specific cluster types is considerably less pronounced than its effect on the cloud capacity planning data set in [15]. The reason is the different structure of the item and bin sizes in the data sets and their relation. While one larger bin offers significantly more packing options than two smaller bins in the cloud capacity planning data set, which compensates for the higher costs, this is not the case for the VCSCVBP data set. Nevertheless, specifically for cost structure (b), the extension improves ΔZ. Considering all cost structures, the Combined CS-P heuristic also outperforms the solver-based solution in terms of runtime, with an average Δt of 3229.8 seconds for the heuristic without extension and of 2496.5 seconds for the heuristic with extension. The VNS improves ΔZ of its initial solution for cost structures (b). The GA for VCSCVBP delivers a quite stable ΔZ between -0.6% and 1.6%, but, on average, is outperformed by the Combined CS-P heuristic without extension in terms of the objective value and runtime.

In summary, we conclude that for the VCSCVBP_HO data set, tailored constructive heuristics such as the Combined CS-P heuristic lead to a high solution quality and shorter runtimes compared to the exact solver-based method. For

Table 4. Comparison of the performance of the Combined CS-P heuristic, VNS, and GA with that of the exact solver-based solution method by instance class on the VCSCVBP_HO data set

Class	Combined CS-P				VNS		GA		Solver	
	Without ext.		With ext.							
	ΔZ (%)	Δt (s)	ΔZ (%)	Δt (s)	ΔZ (%)	Δt (s)	ΔZ (%)	Δt (s)	MIP gap (%)	No. instances terminated[1]
B1_a	−0.9	3234.0	−1.0	2508.6	−0.9	2279.9	0.9	2958.6	10.2	0
B2_a	−0.9	3217.1	−1.1	2461.2	−0.9	2273.9	0.4	2965.6	10.2	0
B3_a	−1.7	3231.8	−1.9	2510.7	−1.7	2269.6	−0.6	2956.9	9.9	0
B4_a	−0.7	3216.6	−0.9	2465.0	−0.7	2264.4	0.1	2960.4	8.9	0
∅ (a)	−1.1	3224.9	**−1.2**	2486.4	−1.1	2272.0	0.2	2960.4	9.8	0
B1_b	2.1	3245.2	−1.1	2531.7	1.0	2535.7	0.5	2967.1	7.6	0
B2_b	3.0	3229.1	−0.7	2485.3	1.7	2552.6	1.2	2975.7	6.8	0
B3_b	2.8	3248.8	−0.5	2545.9	1.1	2451.2	1.2	2966.4	5.2	0
B4_b	2.3	3234.6	−0.8	2502.7	1.5	2450.0	1.6	2969.3	5.1	0
∅ (b)	2.6	3239.4	**−0.8**	2516.4	1.3	2497.4	1.1	2969.6	6.2	0
B1_c	−1.3	3234.4	−1.3	2510.0	−1.3	2304.0	0.3	2952.1	17.4	0
B2_c	−0.7	3217.8	−0.7	2461.7	−0.7	2286.9	0.7	2958.5	17.3	0
B3_c	−1.1	3231.5	−1.1	2510.1	−1.1	2266.8	0.2	2952.1	17.5	0
B4_c	−1.3	3216.3	−1.3	2465.2	−1.3	2259.0	0.1	2955.7	17.4	0
∅ (c)	**−1.1**	3225.0	**−1.1**	2486.8	**−1.1**	2279.2	0.3	2954.6	17.4	0
∅ total	0.1	3229.8	**−1.0**	2496.5	−0.3	2349.5	0.6	2961.5	11.1	0

[1] Number of instances terminated out of 10 instances

specific cost structures, as cost structure (b), it might be advantageous to add an extension that excludes specific cluster types in the second step.

3.3 Heuristics for Clusters of Heterogeneous Bins

This section compares the Combined CS-P heuristics, VNS, and GA for the problem setting with clusters of heterogeneous bins. With 10 instances per instance class, 120 instances are evaluated. The cost structures and cluster configurations of the constructed data set were deliberately chosen to highlight the advantages and disadvantages of the newly considered algorithms, VNS and GA. For the constructive heuristics, we apply the Combined CS-P heuristic without extension. Identifying cluster types for exclusion is more challenging when clusters of heterogeneous bins are considered, compared to clusters of homogeneous bins discussed in the preceding subsection. For VNS, the Combined CSP-P heuristic without extension is used to generate the initial solution.

Table 5 compares the objective value of the best solution found by the heuristics and the solver. The examined heuristics have significantly lower runtimes than the time limit set for the solver of 3600 seconds. Among them, the Combined CS-P heuristic has the lowest mean runtimes, followed by GA and VNS. For cost structures (d) and (f), the VNS with the initial solution by the Combined CS-P heuristic achieves the lowest ΔZ, but GA remains within the same

range. For cost structure (f), the Combined CS-P heuristic even reaches the same ΔZ with lower runtimes, because the VNS cannot improve the initial solution. For cost structure (e), the benefit of the GA becomes apparent, as it outperforms the other heuristics with an average deviation ΔZ from the objective value of the best solution found by the solver of 0.4% compared to 23.0% for the Combined CS-P heuristic, and 15.0% for the VNS. Both the Combined CS-P heuristic and the VNS for the initial value tend to select the third cluster type *Cl_B_6*, owing to its low c/v-ratio. However, this is the cluster type with the most unbalanced bin types, with eight times bin type 200 x 200 and only twice bin type 1000 x 1000. Consequently, although we can fit large items into the two large bins, we need to open many clusters of this type to accommodate all large items in the large bins, leading to under-utilization of the smaller bins. The randomness inherent to the GA helps mitigate the selection bias of the low c/v-ratio, on which the Combined CS-P heuristic and therefore also the procedure for the initial solution of the VNS rely. Overall, the GA obtains a similar average ΔZ for all cost structures.

Table 5. Comparison of the performance of the Combined CS-P heuristic without extension, VNS, and GA with that of the exact solver-based solution method by instance class on the VCSCVBP_HE data set

Class	Combined CS-P		VNS		GA		Solver	
	ΔZ (%)	Δt (s)	ΔZ (%)	Δt (s)	ΔZ (%)	Δt (s)	MIP gap (%)	No. instances terminated[1]
B1_d	0.4	3106.6	−0.2	2567.2	0.4	2218.7	6.2	1
B2_d	0.6	2858.6	0.0	2273.6	0.8	1989.2	4.1	3
B3_d	0.0	2977.5	−0.2	2195.0	0.4	2085.8	6.0	1
B4_d	−0.1	2959.2	−0.4	2161.7	0.5	2072.5	6.7	1
∅ (d)	0.2	2975.5	**−0.2**	2299.4	0.5	2091.6	5.8	0.2
B1_e	22.7	3257.0	15.7	1540.1	0.3	2371.8	8.1	0
B2_e	22.4	3242.8	15.2	1453.9	0.3	2368.3	7.1	0
B3_e	23.0	3257.9	14.4	1362.5	0.1	2353.9	6.5	0
B4_e	23.7	3242.9	14.6	1412.9	0.7	2357.3	6.2	0
∅ (e)	23.0	3250.2	15.0	1442.4	**0.4**	2362.8	7.0	0.0
B1_f	−0.4	3201.7	−0.4	2645.5	0.6	2307.0	6.8	1
B2_f	−0.5	2558.6	−0.5	1976.9	0.5	1680.0	6.7	3
B3_f	−1.0	3243.5	−1.0	2456.8	0.0	2339.3	6.6	1
B4_f	−1.5	3226.6	−1.5	2444.1	−0.6	2338.8	8.0	1
∅ (f)	**−0.9**	3057.6	**−0.9**	2380.8	0.1	2166.3	7.0	0.2
∅ total	7.4	3094.4	4.9	2132.3	**0.3**	2206.9	6.6	0.1

[1] Number of instances terminated out of 10 instances

Due to the high number of improvements observed during the initial 10 iterations of the VNS for specific cost structures, we conduct additional test runs with the number of iterations increased from 10 to 20. At this point, due to space limitations, we do not provide a detailed presentation of the results in a separate table. For cost structures (d), ΔZ improves from −0.2% to −0.3% with

a decrease in Δt from 2299.4 seconds to 1633.7 seconds. For cost structures (e), ΔZ improves from 15.0% to 13.5% with a decrease in Δt from 1442.4 seconds to 332.6 seconds. This demonstrates that the VNS can find better solutions as the number of iterations increases, but at a higher runtime as a tradeoff. Particularly for cost structure (e), there may be even more opportunities for improvement with a greater number of iterations. However, for cost structure (f), increasing the iterations does not impact the best solution found; this is because the VNS cannot improve the initial solution at all for this cost structure.

Finally, we conclude that the effectiveness of the heuristics heavily depends on the problem setting, including the cluster configurations and the cost structures, as already observed in the computational experiments with clusters of homogeneous bins. The GA demonstrates the most consistent performance across all problem settings, whereas the VNS surpasses it in effectiveness for certain specific problem settings. The objective value of the best solution found by the Combined CS-P heuristic differs only slightly from that of the VNS for a large number of considered instances, but has a significantly lower average runtime.

4 Conclusion and Outlook

In this paper, we propose various heuristics specifically tailored to VCSCVBP. We compare the Combined CS-P heuristic, VNS, and GA for VCSCVBP to an exact solver-based solution method. Additionally, we generate two novel benchmark data sets with 120 instances with clusters of homogeneous bins and 120 instances with clusters of heterogeneous bins for VCSCVBP. In an extensive analysis, we demonstrate the advantages of the proposed heuristics relative to each other and to the solver-based approach. The Combined CS-P heuristic achieves consistently high solution quality across the majority of instances. Although VNS and GA typically require higher computational runtimes than the Combined CS-P heuristic, they demonstrate distinct advantages for specific cost structures, particularly for clusters of heterogeneous bins. By randomly swapping cluster types, these methods effectively escape local minima. Notably, the GA exhibits superior performance for instances where clusters with a low c/v-ratio consist primarily of smaller bins alongside a limited number of larger bins. Future research directions for VCSCVBP heuristics include investigating alternative initial solutions for VNS under specific cost structures. Additionally, the parametrization of the various heuristics can be examined in more detail. In conclusion, this study provides a comprehensive overview of different classes of heuristics for VCSCVBP and highlights their respective advantages.

Acknowledgments. The research was conducted as part of the strategic partnership of SAP SE and KIT. We would like to thank SAP SE for the support.

References

1. Borisovsky, P., Eremeev, A., Panin, A., Sakhno, M.: Temporal bin packing problems with placement constraints: MIP-models and complexity. In: Eremeev, A.,

Khachay, M., Kochetov, Y., Mazalov, V., Pardalos, P. (eds.) Mathematical Optimization Theory and Operations Research. vol. 14766, pp. 157–169. Springer Nature Switzerland, Cham (2024). https://doi.org/10.1007/978-3-031-62792-7_11
2. Borisovsky, P., Fedotova, E.: Genetic algorithm for the variable sized vector bin-packing problem with the limited number of bins. In: Kochetov, Y., Eremeev, A., Khamisov, O., Rettieva, A. (eds.) Mathematical Optimization Theory and Operations Research: Recent Trends, Communications in Computer and Information Science, vol. 1661, pp. 55–67. Springer Nature Switzerland, Cham (2022). https://doi.org/10.1007/978-3-031-16224-4_3
3. Caprara, A., Toth, P.: Lower bounds and algorithms for the 2-dimensional vector packing problem. Discrete Applied Mathematics **111**(3), 231–262 (2001). https://doi.org/10.1016/S0166-218X(00)00267-5
4. Crainic, T.G., Perboli, G., Rei, W., Tadei, R.: Efficient lower bounds and heuristics for the variable cost and size bin packing problem. Computers & Operations Research **38**(11), 1474–1482 (2011). https://doi.org/10.1016/j.cor.2011.01.001
5. Gabay, M., Zaourar, S.: Vector bin packing with heterogeneous bins: Application to the machine reassignment problem. Annals of Operations Research **242**(1), 161–194 (2016). https://doi.org/10.1007/s10479-015-1973-7
6. Haouari, M., Serairi, M.: Heuristics for the variable sized bin-packing problem. Computers & Operations Research **36**(10), 2877–2884 (2009). https://doi.org/10.1016/j.cor.2008.12.016
7. Hemmelmayr, V., Schmid, V., Blum, C.: Variable neighbourhood search for the variable sized bin packing problem. Computers & Operations Research **39**(5), 1097–1108 (2012). https://doi.org/10.1016/j.cor.2011.07.003
8. Heßler, K., Gschwind, T., Irnich, S.: Stabilized branch-and-price algorithms for vector packing problems. European Journal of Operational Research **271**(2), 401–419 (2018). https://doi.org/10.1016/j.ejor.2018.04.047
9. Holland, J.H.: Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. MIT press (1992)
10. Kou, L.T., Markowsky, G.: Multidimensional bin packing algorithms. IBM Journal of Research and Development **21**(5), 443–448 (1977). https://doi.org/10.1147/rd.215.0443
11. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & Operations Research **24**(11), 1097–1100 (1997). https://doi.org/10.1016/S0305-0548(97)00031-2
12. Mommessin, C., Erlebach, T., Shakhlevich, N.V.: Classification and evaluation of the algorithms for vector bin packing. Computers & Operations Research **173**, Article 106860 (2025). https://doi.org/10.1016/j.cor.2024.106860
13. Nagel, L., Popov, N., Süß, T., Wang, Z.: Analysis of heuristics for vector scheduling and vector bin packing. In: Sellmann, M., Tierney, K. (eds.) International Conference on Learning and Intelligent Optimization, vol. 14286, pp. 583–598. Springer Nature Switzerland (2023). https://doi.org/10.1007/978-3-031-44505-7_39
14. Panigrahy, R., Talwar, K., Uyeda, L., Wieder, U.: Heuristics for vector bin packing (2011), https://www.microsoft.com/en-us/research/publication/heuristics-for-vector-bin-packing/
15. Wolf, L., Klos, S., Nickel, S.: Variable cost and size cluster vector bin packing – a model and heuristics for cloud capacity planning. In: Zhang, Y., Hladik, M., Moosaei, H. (eds.) Learning and Intelligent Optimization. vol. 15744, pp. 265–280. Springer Nature Switzerland, Cham (2026). https://doi.org/10.1007/978-3-032-09156-7_18